5.0 Runtime Environment

This chapter describes the software configuration for the COE runtime environment. All software and data, excepting low level components of the bootstrap COE, are packaged as segments. A *segment* is a collection of one or more CSCIs (Computer Software Configuration Items) most conveniently managed as a unit. Segments are constructed to keep related CSCIs together so that functionality may be easily included or excluded.

There are six types of segments corresponding to the different types of components that may be added to a system:

1. **COTS:** A segment totally comprised of vendor software.

2. **Data:** A segment composed of a collection of data files for use by the system or by a collection of segments.

3. **Database:** A segment that is to be installed on a database server under the management of the DBMS, and ownership of the DBA.

4. **Account Group:** A segment which serves as a template for establishing a runtime environment for individual operators.

5. **Software:** A collection of executables and static data which extend the base functionality and environment established by an account group.

6. **Patch:** A segment containing a correction to apply to another segment, whether data or software.

In addition, segments may have attached attributes which serve to further define and classify the segment. There are four segment attributes:

1. **Aggregate:** A collection of segments grouped together and managed as an indivisible unit.

2. **Child:** A segment which is part of an aggregate, but is subordinate to a single segment designated as the parent. An aggregate can have multiple child segments.

3. **COE Component:** A segment which implements functionality contained within the COE, as opposed to a mission application segment.

4. **Parent:** The segment which is part of an aggregate that is considered to be the "root" segment. The parent segment name is the name presented

to an operator as the name of the aggregate. An aggregate can have only one parent segment.

> **Note:** Technically, "aggregate" and "COE component" are attributes of a segment. The terms are often used as if they represent segment types. When discussing aggregate segments or COE component segments, it is implicitly understood that there is an underlying segment type, usually software.

Segment installation is accomplished in a disciplined way through instructions contained in files provided with each segment. These files are called *segment descriptor files* and are contained in a special subdirectory, `SegDescrip`, called the *segment descriptor subdirectory*. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system. The format and contents of the segment descriptor files are the central topic of this chapter.

The segment concept and the strict rules which govern the COE and runtime environment provide several benefits:

¥ *Segment developers are decoupled and isolated from one another.* Segments are self contained within an assigned directory. Developers have maximum freedom within the assigned segment directory, but minimum freedom outside it. This allows multiple developers to work in parallel with seamless integration after development.

¥ *Extensions to the COE are coordinated through automated software tools.* It is not possible to create a single configuration of the COE that meets all possible mission application or site unique requirements. However, the COE tools make it possible to extend a base COE in a carefully controlled way to ensure compatibility and identify segment dependencies and conflicts.

¥ *Compliance verification and installation can be automated.* Standards without automated validation are difficult to use in practice, especially in a program where the system is large and there is a need to coordinate activities from several different contractors, program sponsors, services, and agencies. The COE approach to validation is closely related to software installation so that automation of one directly leads to automation techniques for the other.

¥ *Mission application segments are isolated from the COE.* System integration problems are frequently a result of an undisciplined interaction between software components, or because of tight coupling between components.

> The COE controls interaction through APIs and isolates mission applications from the COE component segments so that failure of one mission application segment is less likely to affect another, or affect the stability of the COE foundation itself.

Principles contained in this chapter are fundamental to the successful operation of the COE, and COE compliance is largely measured by this chapter. Developers are required to adhere to the procedures described herein to ensure that segments can be installed and removed correctly, and that segments do not adversely impact one another.

5.1 New Features

This DII COE release includes a number of improvements over previous COE releases. A list of the more significant improvements is provided here for developers who are already familiar with either the JMCIS or GCCS COE.

Modifications have been made for several reasons:

¥ to allow extension to non-Unix environments,
¥ to generalize the COE concept,
¥ to simplify or clarify certain segment descriptor files,
¥ to further reduce integration problems,
¥ to meet emerging requirements, and
¥ to leverage lessons learned.

The present release is completely backwards compatible with previous JMCIS and GCCS COE releases. Segments presently in use do not need to be modified to work with the features described here. However, certain features from previous releases are now obsolete and support for them will eventually be phased out. The tool `VerifySeg` will issue warnings when run against legacy segments to identify obsolete features. Obsolete features are noted in the subsections which follow in an "Obsolete" box.

¥ COE component segments are defined and installed in a special COE directory

¥ `SegInfo` contains most segment information rather than individual segment descriptor files

¥ Segment executables are stored in a `bin` subdirectory rather than a `progs` subdirectory to conform to commercial practice

¥ Library modules are stored in a `lib` subdirectory rather than a `libs` subdirectory to conform to commercial practice

¥ Segments may reserve space to allow room for growth

¥ Segments may request space on multiple disk partitions

¥ Segments may specify NFS mount points

¥ Segments may request system reboot after installation

¥ Segments may affect the user account creation / deletion process

¥ Segments may perform cleanup operations during the `MakeInstall` process

¥ Segments are automatically compressed by `MakeInstall`

¥ `PostInstall` scripts may prompt operators during segment installation

¥ `COEServices` is extended to include other system services

¥ Icons and Menus are supported

¥ Local and remote segments are supported

¥ Character based interfaces are supported

¥ The COE contains a COTS license manager

¥ Variant definitions are supported

¥ A `Processes` descriptor file is supported

¥ `#ifdef` constructs are supported in segment descriptor files

¥ Unix file permissions and owner are set by the install tools

¥ New tools and extensions are described in an appendix

¥ A PC-based COE is supported and is described in an appendix

¥ Segments may use a boolean OR to specify segment dependencies so that a dependency can be fulfilled by one or more segments

¥ Segments may request temporary disk space for use during the installation process, but which will be deleted when the installation is complete

¥ A new segment type is added to accomodate components that are to be managed by the DBMS

5.2 Disk Directory Layout

This subsection describes the COE approach for a standardized disk directory structure for all segments. A standardized approach is required to prevent two segments from overwriting the same file, creating two different files with the same name, or similar issues that cause integration problems. Unfortunately, such problems are often not discovered until the system is operational in the field.

In the COE approach, each segment is assigned its own unique, self contained subdirectory. A segment is not allowed to directly modify any file or resource it doesn't "own" - that is, outside its assigned directory. Files outside a segment's directory are called *community files*. COE tools coordinate modification of all community files at installation time, while APIs to the segments which own the data are used at runtime.

Figure 5-1 shows the COE directory structure. The root level directory for the COE is `/h`. Underneath `/h`, disk space is organized into the following categories (note the close parallel to segment types):

COTS	segment descriptors for installed COTS products
AcctGrps	templates for establishing a runtime environment context
COE	component segments constituting the COE
data	subdirectory for shared (local and global) data files
Segments	one or more subdirectories for mission application or other segments
USERS	operator home directories with operator specific items such as preferences
TOOLS	collection of useful tools for the development environment

Figure 5-1 does not show other important disk directories, such as the `/etc` directory. The `/etc` directory is one of a family of related directories which contain Unix system files. Other COTS products may require specific directories as well.

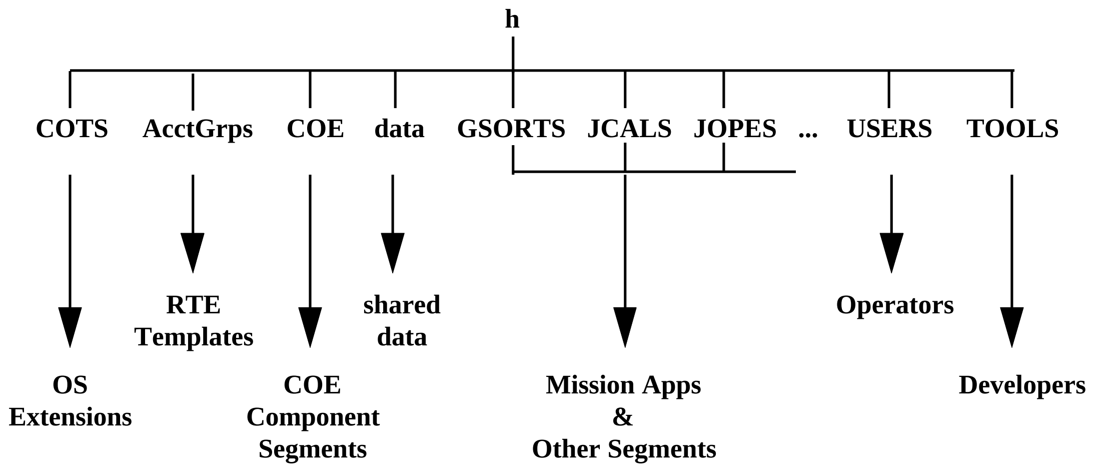

Figure 5-1: COE Directory Structure

Developers may *not* alter or create files outside of their assigned segment directory. COE compliance mandates strict adherence to this directive, with the following exceptions:

1. Temporary files may be placed in the directory pointed to by `TMPDIR` (typically `/tmp`). However, the operating system deletes files in this directory at boot time, and disk space is limited. Developers must use this temporary directory sparingly and shall delete temporary files when an application is done.

2. Segments may place data files in the `/h/data` directory, and are required to do so for shared data (see subsection 5.4.2 below).

3. Operator specific data files shall be placed in subdirectories underneath `/h/USERS` (see subsection 5.2.2 below).

4. Files may be added to the `/h/TOOLS` directory. This is a community directory for tools useful in the development process. Segments shall not place any files in this directory which are required at runtime since this directory is not installed at operational sites. This directory is described in subsection 5.2.3.

5. Segments may request that the COE tools modify community files during the installation process.

6. Segments may issue a request to modify a file to the segment which "owns" the file. This shall be done through use of, and only through use of, published public APIs.

As software is loaded onto the system, the `/h` disk partition may eventually run out of disk space. The installation software will automatically create a symbolic link to preserve the logical structure shown in Figure 5-1, and delete the link when segments are removed. Hence, Figure 5-1 represents a *logical* view, not a *physical* view, of file and directory locations. Due to the potential need to relocate segments at installation time based on available disk space, COE compliant segments must meet the following requirements:

¥ Segments shall use relative pathnames instead of absolute pathnames.

¥ Segments which use symbolic links to point to files contained within the segment shall use relative pathnames for the link.

¥ Segments which use symbolic links to community files may use absolute pathnames as long as (a) the segment can determine the community file's location at install time and (b) the segment can resolve linking to a community file which may itself be a symbolic link.

¥ Segments which add an environment variable to the account group's global runtime environment for locating files within the segment shall use a single "home" environment variable. Environment variables of this nature are normally required only when the segment files are to be accessible by other segments.

To illustrate the last requirement, consider a segment that provides a continuous readout of time-until-impact for a missile. Assume the segment's assigned directory is `MissleTDA` and it's segment prefix is `MSLE`. The `ReqrdScripts` descriptor file (see below) is used to add the following to the account group's `.cshrc` file:

```
setenv MSLE_HOME    /h/MissleTDA
```

`MSLE_HOME` is called the segment's *home environment variable.* Static data within the segment can be referenced by `$MSLE_HOME/data` while executables may be referenced by `$MSLE_HOME/bin`. This technique of using relative pathnames means that segments can be easily relocated at integration or installation time by modifying a single environment variable.

The last requirement stated above does not apply to environment variables defined for use purely within the software development environment. The COE requires that the runtime environment be separated from the development environment. This is typically done by separating environment variables and other settings into physically separate files. The development environment is not present during runtime for the operational system.

Also carefully note that the last requirement stated above applies only to the account group's *global* runtime environment, not a *local* runtime environment. When a segment executable is launched, it inherits the environment established by the account group template. It may then add to its local runtime environment through techniques equivalent to the C `putenv` function.

The time-to-impact example illustrates additional COE requirements regarding definition of a home environment variable.

¥ A segment home environment variable shall point to the segment's home subdirectory, *not* a lower level subdirectory. (e.g., `/h/MissleTDA` and *not* `/h/MissleTDA/Scripts`)

¥ A segment home environment variable, if added to the global environment, shall be added through an environment extension file (see `ReqrdScripts` below).

¥ If a segment home environment variable is required, it shall be named `segprefix_HOME`, where `segprefix` is the segment prefix. Segments which use the same segment prefix must ensure that only one segment defines a home environment variable. This requirement assures that home environment variables are uniquely named between segments.

¥ Segments shall not define a global environment variable that can be derived from an already defined environment variable. For example,

```
setenv MSL_DATA          $MSL_HOME/data
```

is redundant and is therefore not allowed because the expression `$MSL_HOME/data` can be used wherever `$MSL_DATA` can be used.

¥ Segments shall not use the "~" character to specify relative pathnames in the runtime environment, whether to define a home environment variable or any other environment variable.

Unix allows statements of the form

```
source ~/Scripts/.cshrc.tst
```

in `.cshrc`, `.login`, and similar scripts. The "~" character is substituted at run time with the name of the home login directory (as defined in the `/etc/passwd` file). Suppose this statement were contained in a `.cshrc` file and, to prevent making duplicate copies and managing updates to this file, another segment wishes to use the Unix `source` command to include this `.cshrc` file in its own environment. Any segment wishing to source the example `.cshrc` file must duplicate the same disk directory path structure (e.g., must have a `Scripts` subdirectory underneath the home login directory) and must have a file called `.cshrc.tst` underneath the `Scripts` subdirectory. This approach is problematic in the runtime environment because the login home directory is different for every operator, and leads to difficulties in sharing environment settings.

Obsolete: Previous releases of the COE allowed several path related environment variables to be defined in the environment extension file such as

```
     setenv MSL_HOME          /h/MissleTDA
     setenv MSL_DATA          $MSL_HOME/data
     setenv MSL_BIN      $MSL_HOME/bin
```

The Full COE Compliance level limits segments to a single path related environment variable in order to reduce the size of the environment variable space, which is a scare system resource.

5.2.1 Segment Subdirectories

COE compliance mandates specific subdirectories and files underneath a segment directory. These are shown in Figure 5-2 for a general segment. The precise subdirectories and files required depend upon the segment type. For example, a `Scripts` subdirectory is required for account group segments. The `Scripts` subdirectory will normally contain, as a minimum, `.cshrc`, `.xsession`, and `.login` scripts. These serve as a template for establishing a basic runtime environment. For software segments, the `Scripts` subdirectory contains environment extension files.

Some of the subdirectories shown in Figure 5-2 are required only for segment submission and are not delivered to an operational site. Runtime subdirectories normally required are as follows:

`data`	subdirectory for static data items, such as menu items, that are unique to the segment, but will be the same for all users on all workstations
`bin`	executable programs for the segment

`Scripts`	directory containing script files
`SegDescrip`	directory containing segment descriptor files.

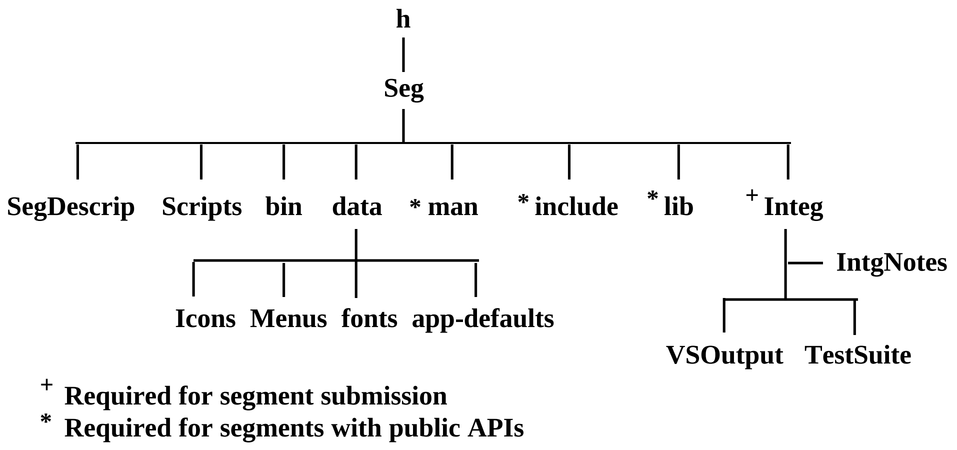

Figure 5-2: Segment Directory Structure

Obsolete: Previous COE releases used a subdirectory named `progs` instead of `bin` to store executables, and a subdirectory `libs` instead of `lib` for storing object libraries. The `progs` and `libs` convention are now obsolete in order to conform with conventional usage. Both `progs` and `libs` will be maintained for backwards compatibility for a short period of time.

The descriptor directory `SegDescrip` is *always* required for *every* segment. Its contents are defined in later subsections. Segment developers may use arbitrary disk file structures during the development phase, but segments shall conform to the structure shown prior to submitting a segment to DISA. It is a violation of the COE to use a different subdirectory name to fulfill the same purpose as any subdirectory shown, or to use a different runtime directory structure than that shown in Figure 5-2.

The distinction between the `Scripts` subdirectory and the `bin` subdirectory is subtle. Files in the `Scripts` subdirectory are used to establish attributes of the runtime environment. Scripts are used here in the sense of traditional Unix, X Windows, or Motif files (`.cshrc`, `.login`, `.xsession`, `.mwmrc`, etc.) that are usually referenced only during the login process, or in the establishment of a separate runtime session. Files of this nature are located in the `Scripts` subdirectory. Executable files may be created as a result of compiling a program, or may be written as a shell. Files of this nature implement executable features of the segment and are located in the `bin` subdirectory.

Subdirectories underneath `data` depend upon whether or not the segment has menu or icon files, or needs additional fonts or app-defaults. During segment installation, special processing is performed on files within the `app-defaults` and `fonts` subdirectories. See subsection 5.4.2 below for more details.

The remaining subdirectories shown in Figure 5-2 are required in order to submit a segment to DISA as follows:

`include`	subdirectory containing header files for public APIs
`lib`	subdirectory containing object code libraries for public APIs
`man`	subdirectory containing Unix "man" pages for public APIs
`Integ`	subdirectory containing items required by the integration process

Segments which do not contain public APIs are not required to submit `include`, `lib`, or `man` subdirectories. For those segments which submit public APIs, private APIs are not allowed in the `include` subdirectory, nor are private libraries allowed in the `lib` subdirectory.

The `Integ` subdirectory serves as a convenient repository for information that needs to be communicated from the developer to the integrator. The file `VSOutput` is *required* for all segments submitted. The subdirectory `TestSuite` is *required* for all segments which submit public APIs and is to contain source code for a program(s) which exercises all APIs submitted. The file `IntgNotes` is *required* for all segments submitted and contains a brief description of why the segment is being submitted (new features, bug fixes, etc.). It also contains any special instructions that needs to be communicated to the integrator for proper segment integration and installation.

5.2.2 USERS Subdirectories

The COE establishes individual operator login accounts and provides a separate subdirectory on the disk for storing operator specific data items. Figure 5-3 shows the directory structure involved. This structure is created and managed automatically as accounts are added or deleted by the Security Administrator software. All users will have a subdirectory underneath `/h/USERS`. The subdirectory name shall have the same name as the login account name. As shown in Figure 5-3, operator accounts may be global or local in scope. A *local* account is workstation specific, whereas *global* accounts are available from any workstation on the LAN.

A special subdirectory, `Profiles`, is shown in Figure 5-3 underneath `/h/USERS/local` and `/h/USERS/global`. This subdirectory contains local and global profile information respectively for establishing what menus, icons, and segments are available to operators belonging to the same profile. A profile is defined for a single account group, but multiple profiles may be defined for the same account group. Operators may thus participate in multiple profiles from multiple account groups.

The file `Segments` underneath the `Profiles` subdirectory is a table listing all segments that are active within the specific profile. The subdirectories `Menus` and `Icons` contain menu and icon files that have been customized to include or exclude segment functions based on how the profile was defined. The `Scripts` subdirectory contains environmental settings that are to be inherited for the given profile. In contrast to previous COE releases, the environment is established, not by all segments loaded on the machine, but only by the active segments for that profile.

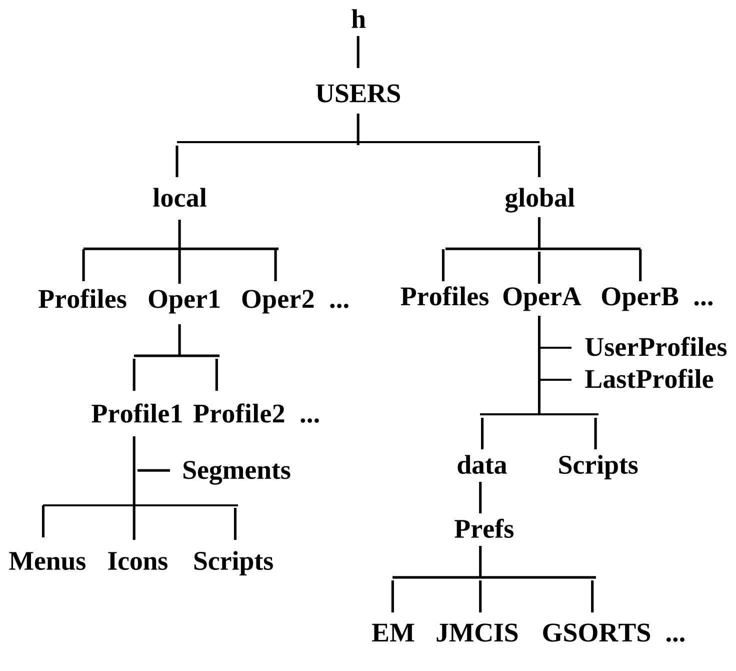

Figure 5-3: Operator Directory Structure

When an operator account is created, certain structures are set up automatically. The file `UserProfiles` is a list of the profiles that the specific operator is authorized to use, and whether the profile is a local or global profile. This approach provides the flexibility of assigning global accounts so that preferences follow the operator, but the actual profile can be locally constrained to a workstation for security or performance reasons. The file `LastProfile` is the profile the operator was in when last logged out. It is an operator convenience feature provided so that the last profile is automatically re-established when the operator logs in again.

The `Scripts` directory created for operator logins serves two purposes. It uses `UserProfiles` and `LastProfile` information to establish the runtime environment context, and it provides a secure fail safe environment in case a profile is unavailable. For example, if a global profile is unavailable because the server is down, an error message is displayed to the user. The user is then either prompted to select another profile from `UserProfiles`, or the operator is logged out if no other profiles are available.

The subdirectory `Prefs` underneath the operator's data directory is used to store segment specific operator preferences. COE compliance requires that segments store all operator preference data here. A segment is responsible for creating its own subdirectory, with the same name as the segment directory, and any required files when the segment first references the preferences data. The exact pathname for the `Prefs` subdirectory will change each time an operator logs in; thus segment software shall use functions from the *Preferences Toolkit APIs* to retrieve the correct pathname for the currently active operator account.

Account group segments define the environment variables `USER_HOME` and `USER_DATA` to point to the correct operator directories. For the example in Figure 5-3, the following assignments would be made:

```
USER_HOME = /h/USERS/global/OperA
USER_DATA = /h/USERS/global/OperA/data
```

Note that `USER_HOME` is *not* defined to be `/h/USERS/global/OperA/Scripts` which is the login home directory.

Segments, such as the Executive Manager, may need to reference menu and icon files for the operator's currently defined profile. However, the directory location for these files is operator dependent and will change during a login session if the operator changes profiles. Segments must use functions contained in the *Preferences Toolkit APIs* to determine the current profile. The environment variable `USER_PROFILE` is also set by the account group segment during login. For example, assume that operator `OperA` in Figure 5-3 has been assigned the local profile `Profile2`. Then the environment variable `USER_PROFILE` is set to

```
USER_PROFILE = /h/USERS/local/Profile2.
```

COE compliance requires adherence to the following:

¥ Segments shall create subdirectories as needed under the operator's `Prefs` subdirectory for storing operator specific data.

¥ Segments shall account for an environment in which accounts are created and deleted, so that a segment will need to create and initialize missing data files that are operator specific.

¥ Account group segments shall set `USER_HOME`, `USER_DATA`, and `USER_PROFILE`. No other segment shall set or alter these environment variables.

¥ Segments shall determine the operator's directory and profile exclusively through the *Preferences Toolkit APIs*, or the environment variables `USER_HOME`, `USER_DATA`, and `USER_PROFILE`.

5.2.3 Developer Subdirectories

Software for the runtime environment is obtained by loading the desired mission application segments and the required COE components. But the development environment is provided separately as a Developer's Toolkit because it is not delivered to, nor required at, an operational site. The Developer's Toolkit includes object code libraries, header files which define the public APIs, and various tools. By convention, tools are loaded underneath the `/h/TOOLS` subdirectory shown in Figure 5-1. This serves as a convenient directory for community contributed software for general development use.

5.3 Segment Prefixes and Reserved Symbols

Each segment is assigned a unique subdirectory underneath `/h` called the *segment's assigned directory*. The assigned directory serves to uniquely identify each segment, but it is too cumbersome for use in naming public symbols. Therefore, each segment is also assigned a 1-6 character alphanumeric string called the *segment prefix*. The segment prefix is used for naming environment variables and in situations, such as public APIs and public libraries, where naming conflicts with other segments must be avoided. All segments shall preface their environment variables with `segprefix_` where `segprefix` is the segment's assigned prefix. For example, the Security Administrator account group segment is assigned the segment prefix `SSO`. All environment variables for this segment are therefore prefaced with the string "`SSO_`".

The segment prefix is also used to uniquely name executables. All COE component segments shall use the segment prefix to name executables, and it is strongly recommended that all segments follow the same convention. This approach simplifies the task of determining the files that go with each segment and reduces the probability of naming conflicts.

Note: Use the segment prefix inside application code in situations where it is important to distinguish one segment from another. For example, when audit information is written to the security audit log, the segment prefix is also written to the audit log to allow determination of which application module generated the auditable event.

It is sometimes convenient for segments to share the same segment prefix. This is true for aggregate segments, or for segments produced by the same contractor. The COE allows segments to share the same segment prefix; however, the burden for avoiding naming conflicts is placed on the segment developer.

The following segment prefixes are reserved:

`CBIF`	Character Based I/F account group segment
`CDE`	Common Desktop Environment segment
`COE`	Common Operating Environment segment
`DBA`	Database Administrator account group segment
`DCE`	Distributed computing environment segment
`DII`	Defense Information Infrastructure segment
`ECEDI`	Electronic Commerce/Electronic Data Interchange segment
`EM`	Executive Manager segment
`GCCS`	Global Command and Control System segment
`GCSS`	Global Command Support System segment

JCALS	Joint Computer-Aided Acquisition and Logistics Support segment
JMCIS	Joint Maritime Command Information System segment
JMTK	Joint Mapping Toolkit segment
MOTIF	Motif
NIPS	Navy NIPS segment
NT	Windows NT segment
ORACLE	Oracle COTS segment
OSS	Navy OSS segment
SA	System Administrator account group segment
SCO	SCO-Unix segment
SSO	Security Administrator account group segment
SYBASE	Sybase COTS segment
TIMS	Navy TIMS segment
UB	Navy Unified Build segment
UNIX	Unix operating system
USER	prefix for operator specific items
WIN	generic Windows segment
WIN95	Windows 95 segment
WINNT	Windows NT segment
XWIN	X Windows

The COE sets five environment variables that must not be confused with the USER prefix or the segment home environment variable.

- The HOME environment variable is set by the operating system to be the login directory; that is, the login directory as contained in the Unix /etc/passwd file. This will normally point to a Scripts subdirectory while the segment "home" environment variable (segprefix_HOME) is usually one level up from HOME.

- The USER environment variable is set by the operating system to be the login account name and does not refer to a directory as does the USER prefix. In most cases, USER_HOME will be /h/USERS/$USER.

- The environment variables LOG_NAME, LOGNAME, and LOGIN_NAME are equivalent to the USER environment variable, but are not always present on every system.

The COE also includes a number of predefined environment variables that are required by Unix, X Windows, or other COTS software. These environment variables are either set automatically by the operating system, or they must be set by an account group segment. Other segments shall not alter these environment variables except as permitted by environment extension files (e.g., extending the `path` environment variable).

`COE_SYS_NAME`	string containing system name (e.g., "GCCS")
*****`DATA_DIR`	`/h/data`
`DISPLAY`	current display surface
`HOME`	user's login directory
`INSTALL_DIR`	absolute pathname for where segment was installed (defined at install time only)
*****`LD_LIBRARY_PATH`	default location of shared X and Motif libraries
*****`LOGNAME`	user's login account name
*****`LOG_NAME`	user's login account name
*****`LOGIN_NAME`	user's login account name
*****`MACHINE`	machine name derived from `uname -m`
*****`MACHINE_CPU`	CPU type derived from `uname -m`
*****`MACHINE_OS`	Operating system derived from `uname -s -r`
`path`	list of paths to search to find an executable
`SHELL`	shell used (e.g., /bin/csh)
`TERM`	terminal type
*****`TMPDIR`	location of the system defined temporary directory
*****`TZ`	time zone information
`USER`	user's login account name
`USER_DATA`	user's data directory under `/h/USERS/local` or `/h/USERS/global`
`USER_HOME`	user's home directory under `/h/USERS/local` or `/h/USERS/global`
`USER_PROFILE`	user's current profile under `/h/USERS/local/Profiles` or `/h/USERS/global/Profiles`
*****`XAPPLRESDIR`	`/h/data/local/app-defaults`
*****`XENVIRONMENT`	`/h/data/local/app-defaults/COEBaseEnv`
*****`XFONTSDIR`	`/h/data/local/fonts`

***** These environment variables are set by the parent COE component segment. The remaining are set by the applicable account group segment.

The COE sets environment variables `MACHINE`, `MACHINE_CPU`, and `MACHINE_OS` to define the hardware and operating system being used. This allows scripts and descriptors to perform operations that are hardware or operating system dependent. Listed below are the possible values set by the COE which may be used as constants in `#ifdef` constructs within descriptor files, or possible values for the appropriate environment variable.

`MACHINE` Environment Variable and Constants

HP	defined for HP platforms running HP-UX
SOL	defined for Sun Sparc workstations running Solaris
SUN	defined for Sun 4 workstations running SunOS

`MACHINE_CPU` Environment Variable and Constants

HP700	defined for HP 700 series workstations
HP712	defined for HP712 workstations
HP715	defined for HP 715 workstations
HP750	defined for HP 750 workstations
HP755	defined for HP 755 workstations
PC386	defined for Intel 80386 workstations
PC486	defined for Intel 80486 workstations
PENTIUM	defined for Intel Pentium workstations
SPARC	defined for Sun Sparc workstations
SUN4	defined for Sun 4 workstations

`MACHINE_OS` Environment Variable and Constants

HPUX	defined for HP-UX workstations
NT	defined for Windows NT workstations
SCO	defined for SCO Unix platforms
SOL	defined for Solaris workstations
SUNOS	defined for SunOS 4.3.x (non-Solaris) workstations
WIN31	defined for Windows 3.1 platforms
WIN95	defined for Windows 95 platforms

Additional Constants

HP	defined for all HP platforms, regardless of OS
PC	defined for all 80x86 platforms, regardless of OS
SPARC	defined for all Sun Sparc workstations, regardless of OS
SUN4	defined for all Sun 4 workstations, regardless of OS

Note that the environment variables (e.g., `MACHINE_CPU`) will have one and only one value, but several constants may be defined for use within the descriptor files. For example, if the hardware platform is an HP715 running HP-UX 9.01, the `MACHINE` environment variable will be set to HP, `MACHINE_CPU` will be set to HP715, `MACHINE_OS` will be set to HPUX, while the constants HP, HP715, HPUX will be defined for use in descriptors.

Obsolete: Earlier versions of the COE used a single environment variable `MACHINE` to specify the hardware and operating system. This has been replaced by two new environment variables, `MACHINE_CPU` and `MACHINE_OS`. `MACHINE` will be maintained for a short period of time to preserve backwards compatibility.

5.4 Segment Types and Attributes

Segment types and attributes were briefly introduced at the beginning of section 5.0. The present subsection describes segment types and attributes in more detail. Segments are the cornerstone of the COE approach. Developers have considerable freedom in building segments. However, there are some important considerations regarding segments.

¥ Creation of an account group segment requires prior approval by the DISA Chief Engineer. Account groups are predefined to establish COE compliant runtime environments.

¥ Creation of a COE component segment requires prior approval by the DISA Chief Engineer.

¥ All COTS products shall be packaged as individual COTS segments. This requirement is mandated to make it easier to handle COTS licenses, and to ensure that a single version of a COTS product is in use. Dependencies on COTS product versions, such as PERL, must be identified and coordinated with DISA to ensure that the proper version is supported by the COE.

¥ Segments shall not modify any file that lies outside the segment's directory. Community files may be modified only through public APIs or through requests made to the COE installation tools.

5.4.1 COTS Segments

The COTS segment type is used to describe the installation of COTS products. If a COTS product can be structured as a software segment, it should be. However, this is usually not possible because where COTS products will be loaded, what environment extensions are required, etc. are often vendor specific.

The COE must retain segment information about all segments, including COTS products. The segment descriptor information for all COTS products is located underneath the directory `/h/COTS` as shown in Figure 5-4. COTS software is not actually stored in the directory `/h/COTS`, only the segment descriptor information, because the actual location of COTS products is often spread across several subdirectories (such as `/usr`, `/usr/lib/X11`, and `/etc`). Figure 5-4 shows the segment descriptor information for the operating system (`UNIX`), the X Windows environment (`XWindows`), and the Motif window manager and libraries (`Motif`). These three subdirectories, along with the actual COTS software, are loaded with the kernel COE.

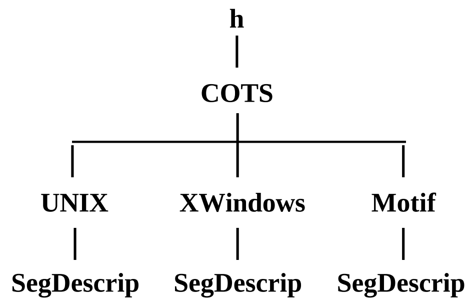

Figure 5-4: COTS Directory Structure

The general approach to handling COTS segments is to create a temporary segment structure in which to store the COTS product, copy the COTS files to their proper location during installation, and then copy the segment descriptor information to `/h/COTS`. It is the responsibility of the `PostInstall` script (see below) to copy the COTS files to their appropriate directories and to perform any other required initialization steps. The installation software handles moving the segment descriptor information to the standard location, `/h/COTS`.

For example, assume a COTS product called `SampleCots` is to be installed which requires loading a series of files into `/etc` (files `f1`, `f2`, and `f3`), `/usr/local` (files `f4` and `f5`), and `/usr/lib` (files `f6`, `f7`, `f8`, and `f9`). A segment directory structure can be set up in whatever manner is most convenient. Figure 5-5 shows one possible solution. The installation software will load the segment `SampleCots` wherever there is room on the disk, and will set the environment variable `INSTALL_DIR` to the absolute pathname for where `SampleCots` was loaded. The `PostInstall` script for this example must recursively copy the subdirectories `etc` and `usr` from `INSTALL_DIR` to `/etc` and `/usr`. The installation software will copy the segment descriptor information to `/h/COTS/SampleCots` and then delete all files underneath `INSTALL_DIR`.

As an alternative, the COE allows a segment to specify exactly where it must be loaded. This is done with the `$HOME_DIR` directive described in subsection 5.5 below. This reduces the need to copy files from one directory to another, and reduces the disk space required during installation.

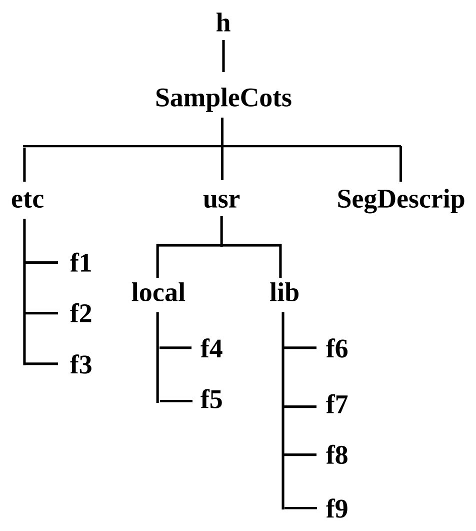

Figure 5-5: Example COTS Segment Structure

The segment descriptor file `FilesList` (see below) is used to document where a COTS product was installed. The `FilesList` descriptor for this example is

```
$PATH:/etc
$FILES
f1
f2
f3
$PATH:/usr
$FILES
local/f4
local/f5
lib/f6
lib/f7
lib/f8
lib/f9
```

To summarize the COTS segment type:

¥ COTS products should be installed as a software segment type if possible.

¥ The COTS segment's `PostInstall` script is responsible for copying files to their required location. The `PostInstall` script must ensure that enough space exists.

¥ The installation software places the segment descriptor information underneath `/h/COTS/name` where `name` is the segment directory name chosen for the temporary segment structure (`SampleCots` in the example above).

¥ The installation software automatically deletes the temporary segment structure after installation is complete.

¥ COTS segments shall document what files are loaded and their location in the `FilesList` segment descriptor file.

5.4.2 Data Segments

Data files are most often created explicitly at runtime by a segment, or loaded as part of the segment itself. However, the ability to load data as a separate segment is useful when there is classified data, optional data, or data that may not be releasable to all communities. The COE supports five categories of data grouped according to data scope, how the data is accessed, and where the data is located:

Global	Data in this category means that every workstation, every application, and every operator on the LAN accesses and uses exactly the same data. Global data is made available through NFS mount points. Examples of global data include the track database and message logs. Global data is located in subdirectories underneath `/h/data/global`.
Database	This category is similar to global data, but access to the data is through a relational database manager. Examples of this kind of data include intelligence databases, JOPES data, and TPFDD files. Data is stored in a database server and is its own segment type.
Local	Local data is limited in scope to an individual workstation. All workstation users and applications access the same data, but the data may differ, and frequently will, from one workstation to another. Examples include overlays, pimtracks, and briefing slides, although the COE provides techniques for exporting these to other workstations. Local data is located in subdirectories underneath `/h/data/local`.
Segment	Segment data is local to a workstation, but is managed and accessed by a single software segment. This data is located under the segment's `data` subdirectory and is typically static data used for segment initialization.
Operator	Data in this category is specific to an operator and is the most limited in scope. Typical examples include preferences for

map colors, location of various windows, and font size. Operator data is stored in a `data` subdirectory underneath `/h/USERS` created for the operator when the operator login account is created, as described in subsection 5.2.2.

There are some important considerations with respect to these data categories:

¥ Data is not necessarily available to an operator or process even if the data scope would otherwise permit it. Discretionary access controls within the COE limit access based upon the security policy of the system.

¥ In some cases, data that could be global is replicated on every workstation to improve system performance. For example, World Vector Shoreline data is identical for everyone on the LAN, and hence meets the criteria for the global data category. However, for efficiency, this data is replicated on each workstation which requires maps and is thus considered local.

¥ Distinction is made between segment data and local data because it affects where the data is stored on the disk. Local data is stored in a single place to make it easier for doing data backups. Because segment data is normally static, it does not need to be archived.

Segment data created at runtime or loaded as part of the segment does not require any special consideration by the COE. The remainder of this subsection will deal with the COE requirements for local and global data, and then present an example of how a data segment is structured for local, global, and segment scope.

global and local

Figure 5-6 shows the directory structure for global and local data. The COE runtime environment sets the environment variable `DATA_DIR` to point to `/h/data`. Segments shall use this environment variable to reference global or local data. The segment which owns the local or global data is responsible for creating and managing its data subdirectories underneath `/h/data/local` and `/h/data/global`. Assuming the segment's assigned directory is `SegDir`, the segment shall create a subdirectory of the form `SegDir/data`.

For example, suppose a segment that does ASW planning is located underneath `/h/ASW` and it will create both global and local data. Then the ASW segment must create subdirectory `/h/data/local/ASW/data` for local data and subdirectory `/h/data/global/ASW/data` for global data.

The COE mandates that local and global data be structured in this fashion for the following reasons:

¥ Centralizing data makes it easier to archive and restore. A simple data archive/restore utility can be created without needing to know how many segments are loaded in the system.

¥ Separating data from software makes it simple to load the software without destructively overwriting existing data. This is especially important as segments are upgraded.

¥ Collecting all global data under a single directory reduces the number of NFS mount points and improves overall network performance.

¥ Organizing data into a standard structure simplifies training and simplifies determination of what data is loaded in the system.

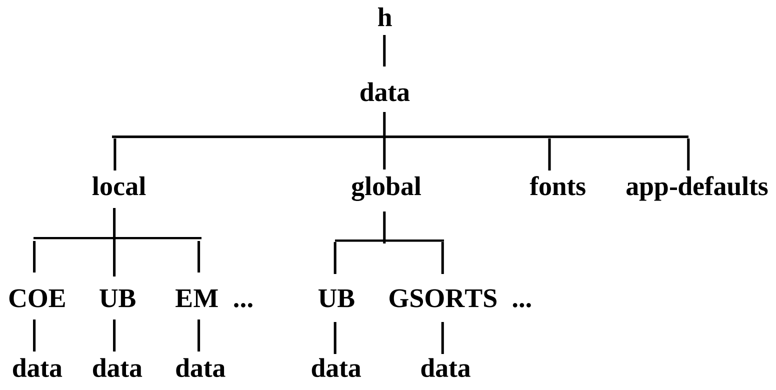

Figure 5-6: Data Directory Structure

fonts and app-defaults

Figure 5-6 shows two additional subdirectories, `fonts` and `app-defaults`. The COE sets environment variables `XFONTSDIR` and `XAPPLRESDIR` to point to these subdirectories. Their purpose is to contain additional fonts (such as NTDS symbology) or application resource files that are not provided by the standard X/Motif distribution. It is a violation of the COE for a segment to overwrite or add files to the standard X/Motif distribution.

During installation, the installation tools look for subdirectories `data/fonts` and `data/app-defaults` underneath the segment's directory. Files contained within these subdirectories must use the segment prefix to guarantee unique names. The installation tools creates a symbolic link underneath the directory `/h/data/fonts` to every file in the segment's `data/fonts` subdirectory, and removes the links when the segment is deinstalled. Similarly, links are created for files underneath the segment's `data/app-defaults` subdirectory.

Creating a data segment requires additional considerations. A segment structure is created for the data and the installation tools *logically* insert the data underneath `/h/data` for global and local scope, but underneath the parent segment for segment data. This is best described through use of an example.

Assume a mine counter measures decision aid has an assigned directory of `MineTDA`. Assume that a separate data segment is to contain parametric data on floating, proximity, and land mines for the decision aid. Figure 5-7 shows the appropriate directory structure for the data segment. Further assume that when installed, the decision aid is located underneath `/h/MineTDA`. Consider how the installation tools handle the mine data segment for global, local, and segment scope.

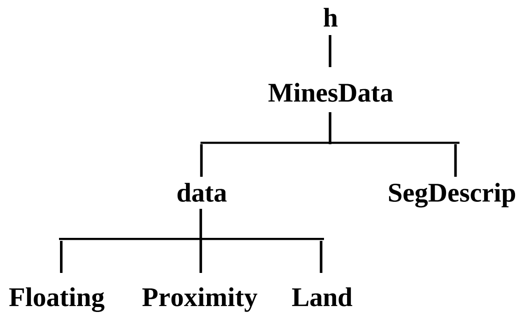

Figure 5-7: Example Data Segment Structure

`global`

The `Data` descriptor file (see below) describe the data scope. For a global data segment, the installation tools will load the mine data underneath `/h/data/global/MineData`. If there is insufficient space to load the segment underneath `/h/data/global`, the install tools will report an error and abort. The mine TDA can thus reference global proximity mine data as being underneath `$DATA_DIR/global/MineData/data/Proximity`.

`local`

For a local data segment, the installation tools will load the mine data on the first available disk partition. The installation tools will then create a symbolic link from `/h/data/local/MineData/data` to wherever the data segment was actually loaded. That is, if the data segment is loaded underneath `/home2/MineData`, then the symbolic link will point to the directory `/home2/MineData/data`. The mine TDA can reference local proximity mine data as being underneath `$DATA_DIR/local/MineData/data/Proximity`.

segment

For segment scope data, the installation tools will load the mine data on the first available disk partition. A symbolic link is then created from the directory `/h/MineTDA/data/MineData/data` to wherever the data segment was actually loaded. Proximity data can thus be referenced as being underneath `$HOME_DIR/data/MineData/data/Proximity`.

It should now be clear why the COE requires that segments which dynamically create global or local data do so underneath a directory of the form `SegDir/data`, where `SegDir` is the name of the segment's assigned directory. This creates a uniform technique for locating files whether they are created directly by a segment, or loaded as part of a data segment.

In summary, COE compliance mandates that:

¥ Segments shall create a data subdirectory underneath `/h/data` for global and local data if they own global or local data. The subdirectory created shall be `SegDir/data` where `SegDir` is the name of the segment's assigned directory.

¥ The parent COE component segment shall set the environment variable `DATA_DIR` to point to `/h/data`.

¥ The parent COE component segment shall set environment variables `XFONTSDIR` and `XAPPLRESDIR` to point to `/h/data/local/fonts` and `/h/data/local/app-defaults` respectively.

¥ Segments shall use the environment variable DATA_DIR to reference data underneath /h/data.

¥ Segments shall account for the initial creation of the segment's data subdirectories underneath `/h/data`.

¥ Segments shall account for proper initialization in the event a data file is not present or is corrupted.

¥ Segments shall place fonts that need to be accessible via `XFONTSDIR` in the segment's `data/fonts` subdirectory. Files in this subdirectory shall be named using the segment prefix.

¥ Segments shall place application resource files that need to be accessible via `XAPPLRESDIR` in the segment's `data/app-defaults` subdirectory. Files in this subdirectory shall be named using the segment prefix.

5.4.3 Database Segment

A database segment contains everything that is to be installed on the database server under the management of the DBMS and the ownership of the DBA. It contains the component database and any utilities provided by the developers for the DBA's use in installing and filling that particular database. Database segments may only be installed on a database server.

When a database segment is installed it must first lay down any scripts, data files, etc. that will be used to create the database. These scripts are then executed by the `PostInstall` process to create the component database. They must first allocate storage to hold the database, and create one or more database accounts to own that database. They then can create the database, within the storage just allocated, and fill it with data. Finally, roles are created to manage access, and the roles are given the appropriate privileges.

Developers cannot provide data files for the DBMS as part of the segment. Database files must be created using the DBMS vendor's utilities (e.g. Oracle'sSQL*DBA 'CREATE TABLESPACE' command) to be correctly incorporated in the DBMS instance.

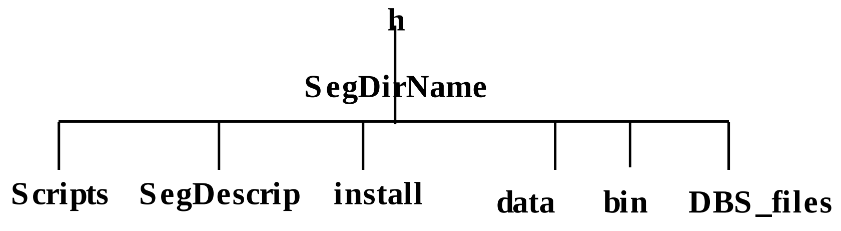

Figure 5-8: Database Segment Structure

Figure 5-8 shows the directory structure required for a database segment. `SegDirName` is the segment's assigned directory. It must be unique and it must be the same as the name of the database owner account for the segment's data objects.

`Scripts`

The `Scripts` directory shall contains any segment specific scripts needed to set the runtime environment for the database installation. This will include environment variables for all directory paths that are referenced by the installation scripts.

SegDescrip Directory

The SegDescrip directory contains the descriptor files necessary to install the database segment. The PreInstall descriptor should prompt the installer to provide the password for the DBMS' database administrator account. The ReleaseNotes descriptor should show how applications, operating system groups and database roles are associated. Developers should also provide the database schema including its dependencies. In addition to any narrative information in this file, developers should include comments on their schema, data objects and data elements as part of their database build.

install

The install directory contains the scripts to install and then create the database segment. It includes all of the database definition language (DDL) scripts that create the database objects for the segment. There are two sets of DDL scripts in this directory. The first set allocates storage for the database, creates the database owner, and defines the roles associated with the database segment. It must be executed by a DBA. The second set creates all database objects (tables, views, indexes, sequences, constraints, and triggers) that make up the database. This set must be executed by the database owner.

data

The data directory contains any data files used to load the database. Data fill may also be provided in a separate data segment if developers wish or need to keep the fill separate.

bin

The bin directory contains any scripts or other executables used to load data from the data files into the database.

DBS_files

The DBS_files directory contains the DBMS-controlled data files that make up the storage for the database. This directory is owned by the DBMS, not the segment. The data files are created during the installation of the segment, normally in the PostInstall process. Directory ownership must be transferred to the DBMS before the data files are created.

5.4.4 Account Group Segments

An account group segment is a template for establishing a basic runtime environment context that other segments may extend in a controlled fashion. An account group segment determines

¥ the processes to launch,
¥ the order in which to launch processes, and
¥ the required environment script files (`.cshrc`, `.xsession`, etc.).

Account groups may also contain executables and data in the subdirectories identified in Figure 5-2.

GCCS contains five predefined account groups. They are located underneath `/h/AcctGrps` shown in Figure 5-1. The predefined account groups, listed by their assigned directory, are as follows:

`CharIF`	account group for character based interfaces
`DBAdm`	account group for database administrators
`GCCS`	account group for normal mission applications
`SecAdm`	account group for security administrators
`SysAdm`	account group for system administrators

COE-based systems will substitute their own account group for `GCCS` (e.g., `GCSS` for the Global Command Support System). They will include `CharIF` if the system supports a character based interface, and may include other account groups to suit system mission requirements.

Figure 5-9 shows how the System Administrator account group is structured. The structure of other account group segments is the same.

`bin`

Account groups utilize COE executables, located underneath `/h/COE/bin`, but will usually include additional account group specific programs. These are located in the account group's `bin` subdirectory. COE compliance requires that executables within this subdirectory use the segment prefix to avoid potential naming conflicts with other executables.

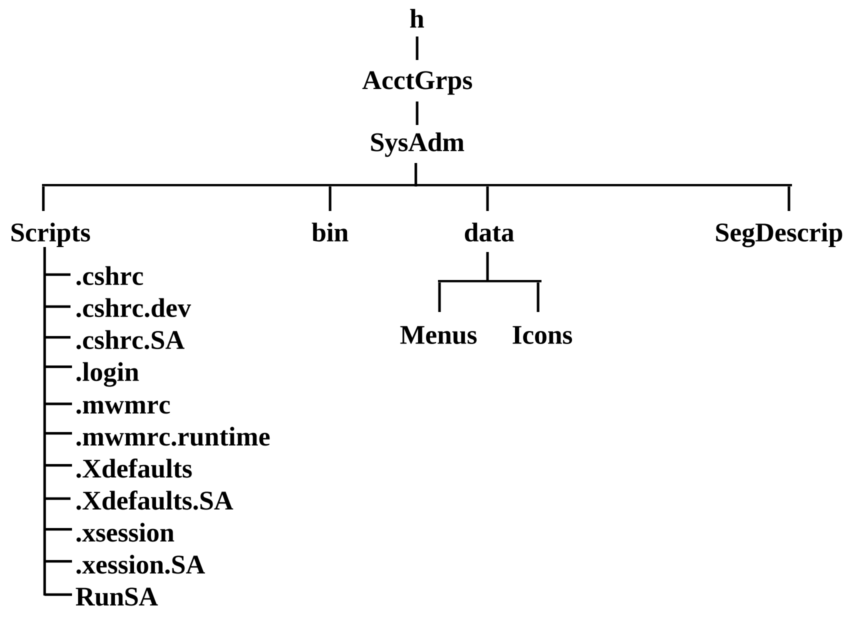

Figure 5-9: Example Account Group Directory Structure

`data`

Segment data specific to the System Administrator account group is located in the `data` subdirectory. The `Menus` subdirectory contains menu files that have menu entries for all options available from the basic System Administrator application. The installation software may modify files contained here to add other menu options. Account group menu files are used as templates from which profiles are created by including or excluding desired menu items. The `Icons` subdirectory is analogous, but defines icons for use by the desktop for launching applications.

`Scripts`

An account group segment will usually contain the following to establish the runtime environment:

`.cshrc`	define environment variables
`.login`	define terminal characteristics
`.mwmrc`	define keyboard and mouse button bindings
`.Xdefaults`	define Motif "look and feel" parameters
`.xsession`	define operations to perform upon initiating an X Windows session.

Precise contents of these files is application dependent. After logging in, Unix first executes the `.cshrc` file, then the `.xsession` file, then the `.login` file to establish the runtime environment context.

Other segments may be loaded to extend the environment established by the account group. This is done through *environment extension files*. COE-compliant account group segments shall name extension files in the form

```
scriptname.segprefix
```

where `scriptname` is the environment file to be extended and `segprefix` is the segment prefix. For the example shown in Figure 5-9, the environment extension files are:

`.cshrc.SA`	extensions to the `.cshrc` file
`.login.SA`	extensions to the `.login` file
`.Xdefaults.SA`	extensions to the `.Xdefaults` file
`.xsession.SA`	extensions to the `.xsession` file.

The `.mwmrc` file does not need to be extended, and extension of the `.login` file is rarely required.

Environment extension files permit COE installation software to provide segment specific environment modifications. A segment uses the descriptor file `ReqrdScripts` (see below) to indicate which environment file to extend, and the installation tools modify the proper file within the account group segment.

For example, suppose the installation tools have loaded a segment underneath `/h/SAOpt`, and the `SAOpt` segment has an environment extension file named `.cshrc.SAOpt` in the segment's `Scripts` subdirectory. The installation tools will include the new environment settings by inserting the following statements in the account group's file `.cshrc.SA`:

```
if (-e /h/SAOpt/Scripts/.cshrc.SAOpt) then
     source /h/SAOpt/Scripts/.cshrc.SAOpt
endif
```

The installation tools automatically remove these statements from `.cshrc.SA` if the segment `SAOpt` is deleted.

Account group segment developer's shall ensure that environment files are included in the account group segment's script files. The `.cshrc` file example shown in Figure 5-9 includes the following statements

```
if (-e $SA_HOME/Scripts/.cshrc.SA) then
```

```
          source $SA_HOME/Scripts/.cshrc.SA
     endif
```

to account for .cshrc extensions. Also note that the source command shall be of the form

```
     source $SA_HOME/Scripts/.cshrc.SA
```

rather than

```
     source $USER_HOME/Scripts/.cshrc.SA
```

The COE mandated form permits a single copy of the environment extension file that must be updated and maintained by the installation software.

The files .cshrc.dev and .mwmrc.runtime shown in Figure 5-9 relate to the software development environment. Developer preferences such as alias commands are included in .cshrc.dev. These preferences shall not be included as part of the runtime environment. A technique such as

```
     if ($?DEVELOPER) then
          source $SA_HOME/Scripts/.cshrc.dev
     endif
```

within the .cshrc file is required to achieve separation of the development environment from the runtime environment.

This technique will not work for certain files, such as .mwmrc, because they do not support conditional statements. The COE tools that create operator accounts explicitly look for script files with an extension of .runtime and uses them to create the operator's runtime environment. In the example shown in Figure 5-9, the account creation tools will copy the file .mwmrc.runtime into the operator's Scripts subdirectory and rename the copy as .mwmrc.

Account groups must also include the base environment established by the COE. Subsection 5.4.8 below describes the COE component segments in more detail. The .cshrc file in Figure 5-9 includes the base COE environment with the statements

```
     if (-e /h/COE/Scripts/.cshrc.COE) then
          source /h/COE/Scripts/.cshrc.COE
     endif
```

The remaining files in Figure 5-9 contain similar statements to include other COE environmental settings.

Account groups must also provide a script or program which launches the application. This is the file named `RunSA` shown in Figure 5-9. COE compliance requires this file to be located underneath the `Scripts` subdirectory.

To summarize compliance requirements for account groups:

¥ Account group segments shall provide environment extension files of the form `filename.segprefix`, where `segprefix` is the account group's segment prefix, for any files that other segments may extend (e.g., `.cshrc.SA` for the `SysAdm` account group).

¥ Account group executables shall use the segment prefix to avoid naming conflicts.

¥ Account group segments shall not include the developer environment as part of the runtime environment.

¥ Account group segments shall provide a single program or script with the name `Runxxx`, where `xxx` is the segment prefix, to initiate execution of the account group's application. This executable shall be located in the account group segment's `Scripts` subdirectory.

¥ Account group segments shall automatically include environment settings established in `/h/COE/Scripts`.

¥ Segment developers shall not modify account group files except through use of the installation software.

¥ Segment developers shall not override environmental settings established by the account group. Segments may use environment extension files to expand the environmental settings.

5.4.5 Software Segments

Software segments add functionality to one or more account groups. The account group(s) to which the software segment applies is called the *affected account group(s)*. The directory structure for a software segment was presented in Figure 5-2.

Software segments frequently need to extend the runtime environment, add new menus and icons to the desktop, and include new executables in the search path. Environment extension files are located underneath the software segment's `Scripts` subdirectory. The `ReqrdScripts` segment descriptor file (see below) indicate which environment files are to be extended.

Software segments provide additional menu and icon files underneath the segment's `data/Menus` and `data/Icons` subdirectories respectively. The segment descriptor files `Menus` and `Icons` (see below) are used to describe where the new items are to appear on the desktop. At installation time, the menu and icon files from all contributing segments are added to the affected account group. This then serves as a master template of all possible functions provided within the account group. Profiles are then created by selectively including or excluding functions within this master template.

Segments that provide executables must ensure that the `bin` subdirectory is included in the search path. This is accomplished by including a statement of the following form in a `.cshrc` extension file:

```
set path=($path $segprefix_HOME/bin)
```

The segment shall append its `bin` subdirectory at the *end* of the search path, *not* the beginning. An implied aspect of this requirement is that segments can not depend upon a specific loading sequence, other than that a segment will not be loaded until after all segments it depends upon are loaded.

COE compliance requires the following:

¥ Segments shall not make separate copies of executables from other segments, the operating system, or other COTS products.

¥ Segments shall use environment extension files as necessary to extend the environment established by the affected account group.

¥ Segments shall use the segment prefix to name objects whenever conflicts may arise with other segments.

¥ Segments shall be completely self contained. Dependencies on, or conflicts with, other segments shall be specified through the appropriate `Requires` or `Conflicts` segment descriptor files.

¥ Segments shall include their `bin` subdirectory at the end of the search path, not the beginning.

5.4.6 Patch Segments

The COE supports the ability to install field patches on an installed software base. A patch segment means the replacement of a collection of one or more individual files, including those of the operating system. It does *not* refer to overwriting a portion of a file, as is sometimes done to patch a section of binary code.

Patches are created in a segment whose directory name is the name of the affected segment followed by a ".", followed by the letter "P", followed by the patch number. Figure 5-10 shows an example patch segment directory structure for applying patch 5 to an ASW segment. The subdirectory `SegDescrip` is required, but the remaining subdirectories are patch dependent. The example illustrates a situation in which scripts, executables, and data files are to be replaced by a single patch.

The installation software loads patches underneath the affected segment in a subdirectory called `Patches`. Figure 5-11 shows the result of loading patch 5 from Figure 5-10. This approach makes it easy to find and identify what patches have been applied to a segment. It also makes it easy for the installation software to automatically remove patches when a segment is replaced by a later update. If there is insufficient room to physically load the patch underneath the `Patches` subdirectory, the patch is loaded on the first available disk partition. A symbolic link is then created to preserve the logical structure shown. Also note that when installed, the resulting subdirectory name of the patch for this example is `P5`, not `ASW.P5`.

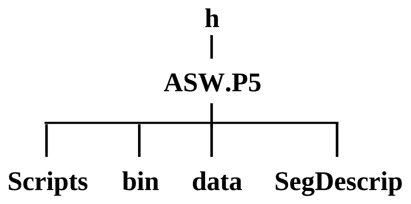

Figure 5-10: Example Patch Directory Structure

As patches are installed and deinstalled, the descriptor file `Installed` in the descriptor directory for the affected segment is updated to reflect what patches are installed and removed, the date and time, the installer's name, and the workstation from which the work was done.

When a patch is installed, it is the patch segment's responsibility to perform whatever operations are necessary to replace files. In the example shown, the `PostInstall` script must copy files from `Scripts`, `bin`, and `data` as required to update files in the existing ASW segment.

To facilitate patch removal, the `PostInstall` program may create compressed copies of files before they are modified and put them underneath the patch subdirectory (e.g., the `ASW/Patches/P5` subdirectory in this example). In this way, a `DEINSTALL` descriptor simply needs to copy the files from the patch subdirectory to their original place and decompress them to restore the system to the pre-patch state. If the files being replaced are large, this may require too much disk space to store the original files. In such cases, the patch segment should be designated as a permanent patch and not make copies. A patch segment is considered to be permanent if the patch segment does not include a `DEINSTALL` descriptor.

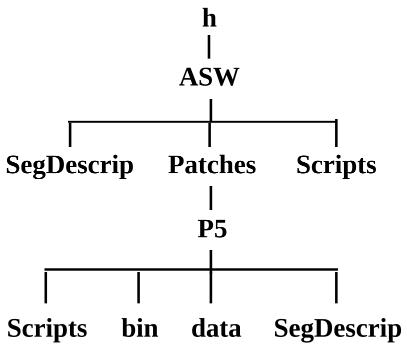

Figure 5-11: Example Installed Patch

The COE installation software assumes that higher numbered patches must be removed before a lower numbered patch can be removed. For example, patch 2 can not be removed until patch 5 is removed. However, if patch 5 can not be removed - because there is no `DEINSTALL` descriptor for patch 5 - patches 1 and 2 can not be removed. The only way to remove them is to remove the entire segment.

COE compliance requires that:

¥ Patch segments shall be named `SegDir.Pnumber` where `SegDir` is the directory name of the segment to be patched, and `number` is a sequential patch number.

¥ Patch segments shall perform the necessary operations to replace files through the `PostInstall` script.

¥ Permanent patch segments shall be designated by the absence of a `DEINSTALL` script.

5.4.7 Aggregate Attribute

It is sometimes convenient for a collection of segments to be treated as an indivisible unit. The aggregate attribute provides this capability, and the collection of segments are called an aggregate segment. One, and only one, segment is designated as the *parent* segment and the remaining segments are designated as *children*. Parent and child segments are designated as members of an aggregate in the `SegType` descriptor file. The child segment must list its parent segment, while the parent segment must list each child in the aggregate. See subsection 5.5 below for the segment descriptor information required to do this. Each segment within the aggregate is packaged according to its segment type as described in preceding subsections.

The parent segment plays a special role in the aggregate. During installation, only the parent segment is "seen" by the operator. Child components are not displayed as selectable items, but are automatically loaded with the parent. Therefore, the segment name and release notes associated with the parent segment should be carefully chosen to be properly descriptive of the aggregate.

The parent segment is the first segment loaded from the aggregate. Child segments are loaded next in the order listed by the parent segment. Because of this, child segments may specify a dependency on the parent, but shall not specify dependencies upon one another.

The COE requires that each segment include a `Security` segment descriptor file. The security level of the parent segment must dominate that of the child segments. The segment developer must ensure that each segment in the aggregate is compatible for the hardware platform. `VerifySeg` will check for this condition and reject an aggregate with incompatible hardware platforms specified.

Disk space required is specified by each individual segment, not by the aggregate parent. The COE installation tools may load parent and child segments on different disk partitions, depending upon space available at install time.

COE compliance requires:

¥ One and only segment in the aggregate shall be designated as the parent segment.

¥ Child segments may specify a dependency on the parent, but shall not specify dependencies upon one another.

¥ The security level of the parent segment shall dominate the security level of all child segments.

¥ Segments within an aggregate shall be consistent with regards to the hardware platform specified.

¥ Segments shall individually specify their own disk space requirements.

5.4.8 COE Component Attribute

Authorized segments may specify the attribute of being a COE component segment. COE component segments are similar to aggregate segments in that one segment serves the role of a parent segment and all others are children to that parent. The parent segment is similar to an account group segment which is affected by a collection of child component segments. However, there are important differences between COE component segments and aggregate segments, and between the parent COE component segment and account groups.

¥ At installation time, a segment identified as a COE component is compared against a table containing the names of authorized COE component segments. If it does not match, the segment is rejected.

¥ Exactly one segment is designated as the parent COE component for the entire system. This is the segment whose directory is `/h/COE`.

¥ Child COE component segments are not loaded unless they are required.

¥ COE component segments are organized into a very specific structure.

¥ The parent COE component segment does not set up a runtime environment. It sets up a baseline environment which is inherited by all account groups.

¥ The parent COE component segment, since it is not an account group, does not require a `RunCOE` file.

Figure 5-12 shows the directory structure for COE component segments. Since COE components form the foundation for the entire system, they are collected together in a single place and are validated more rigorously during segment development, integration, and installation. Special processing, as explained below, is performed on the COE components because of their unique position within the architecture.

The `SegDescrip` subdirectory, required for all segments, underneath `/h/COE` refers to the collection of COE components as a whole. Segments designated as child COE components are loaded in the subdirectory `/h/COE/Comp`. Each child

COE component segment has its own `SegDescrip`, `bin`, `Scripts`, and `data` subdirectory as appropriate. If insufficient space exists to load the COE component directly under `/h/COE/Comp`, a symbolic link is created to where the segment was actually loaded.

Environment files underneath `/h/COE/Scripts` are included by every account group so that they are automatically inherited by every segment. The file `.cshrc.COE` sets the path environment variable so that `/h/COE/bin` is first in the search path before any other segments. Environment extensions for child COE components are handled differently than environment extensions for other segments. As child COE component segments are installed, environment extension files located underneath the child COE component's `Scripts` subdirectory are textually inserted directly into the appropriate file underneath `/h/COE/Scripts`. This insertion is performed automatically by the installation tools. This is done to avoid the runtime overhead of executing several `source` statements to pick up child segment extensions..

Child COE component segments shall not alter the `path` environment variable. It is not necessary to do so because as child COE components are loaded, the installation tools create a symbolic link underneath `/h/COE/bin` to where the executables were actually loaded. This is done so that the search path contains only one entry for the COE, regardless of the number of actual segments comprising the installed COE. This approach mandates that all COE component segments use the segment prefix to name executables. `VerifySeg` will strictly fail COE component segments that do not meet this requirement.

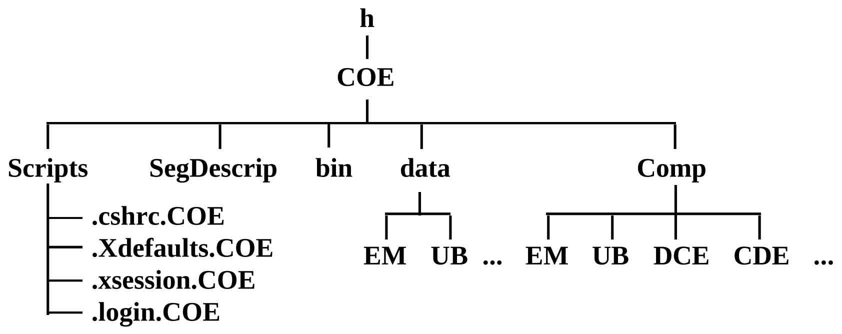

Figure 5-12: COE Directory Structure

Symbolic links are also created underneath `/h/COE/data` to point to the child COE component's `data` subdirectory. The installation tools automatically delete these symbolic links when a COE component segment is deinstalled.

To summarize COE compliance requirements:

¥ COE components shall be authorized by the DISA Chief Engineer.

¥ Child COE components shall not alter the `path` environment variable.

¥ COE components shall use the segment prefix to name all executables.

¥ Child COE components shall use the segment prefix to name all public symbols contained in files within the segment's `Scripts` subdirectory.

5.4.9 Segment Dependencies

Segments specify dependencies upon one another through the `Requires` descriptor described below. However, the COE does not allow circular dependencies. That is, a situation such as Seg A depends upon Seg B, Seg B depends upon Seg C, and Seg C depends upon Seg A is strictly forbidden.

Components of an aggregate may have dependencies upon other components within the same aggregate. But since components of an aggregate are always loaded together as a unit, this does not pose a problem. Components of an aggregate must *not* specify dependencies upon one another in the `Requires` file, even if this is indeed the case. Likewise, the parent segment must *not* specify a dependency on children within the aggregate.

5.5 Segment Descriptors

This section details the contents of the segment descriptor files. These files are the key to providing seamless and coordinated systems integration across all segments. Adherence to the format described here is required for all segments to ensure COE compliance. This enables automatic verification and installation of segments.

The software tool `VerifySeg` must be run during the development phase to ensure that segments properly use segment descriptor files. The software tool `MakeInstall` uses information in segment descriptor files to compress and package segments in a format suitable for installation from tape, from a disk based LAN segment server, from a remote site, or other media. At installation time, the installation tools use segment descriptor information to make the COE changes required (e.g., update menu files) so that software components are available to the user.

Some segment information is contained within individual files while other segment information is collected into a single file, `SegInfo`. Table 5-1 lists each of the descriptor files and which are required, optional, or not applicable for each segment type or segment attribute. Table 5-2 lists the same information for segment descriptor sections within the `SegInfo` descriptor file. The `VerifySeg` tool will display these two tables when the `-t` flag is given on the command line.

`SegInfo` is an ASCII file with multiple sections containing segment descriptor information. A segment section begins with a single line of the form

```
[section name]
```

where *`section name`* is chosen from the list in Table 5-2. A section continues until another section name is encountered, or the end of the file is reached. A section may appear only once within the `SegInfo` file, but the order in which sections appear is unimportant. Section names are not case sensitive.

To preserve backwards compatibility, developers may use individual segment descriptor files rather than sections within `SegInfo`. The `SegInfo` enhancement simplifies the task of creating and maintaining segment descriptor information. The file name required for individual segment descriptors is the same as the section name shown in Table 5-2, but because Unix filenames are case sensitive, the filenames must exactly match the section names shown. A segment may use individual segment descriptor files, or `SegInfo`, but not both.

Obsolete: Individual segment descriptor files is now obsolete and will be phased out. Use SegInfo instead. This is required for the Full COE Compliance Level.

File	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W Seg	Patch
DEINSTALL	O	O	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O	O	O
Installed	I	I	I	I	I	I	I	I
PostInstall	O	O	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R	R	R
SegName	R	R	R	R	R	R	R	R
Validated	I	I	I	I	I	I	I	I
VERSION	R	R	R	R	R	R	R	R

R - Required O - Optional N - Not Applicable
I - Created by Integrator or Installation Software

Table 5-1: Segment Descriptor Files

Section	Acct Grp	Agg	COE Comp	COTS	Data	DB	S/W Seg	Patch
AcctGroup	R	O	N	N	N	N	N	N
COEServices	O	O	O	O	O	O	O	O
Community	O	O	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O	O	O
Compat	O	O	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O	O	O
Data	N	N	N	N	R	N	N	N
Database	X	X	X	X	X	X	X	X
Direct	O	O	O	O	O	O	O	O
FilesList	O	O	O	R	O	O	O	O
Hardware	R	R	R	R	R	R	R	R
Icons	R	O	N	O	N	N	O	O
Menus	R	O	N	O	N	N	O	O
ModName	*	*	*	*	*	*	*	*
ModVerify	*	*	*	*	*	*	*	*
Network	N	N	O	N	N	N	N	N
Permissions	O	O	N	N	N	N	O	O
Processes	O	O	O	O	N	N	O	O
ReqrdScripts	R	O	O	N	N	N	O	N
Requires	O	O	O	O	O	O	O	O
Security	R	R	R	R	R	R	R	R
SegType	*	*	*	*	*	*	*	*

R - Required O - Optional N - Not Applicable
X - Reserved for Future * - Obsolete

Table 5-2: SegInfo Segment Descriptor Sections

Certain general characteristics are common to all files or sections listed in Tables 5-1 and 5-2:

1. All descriptor files are ASCII files.

2. In describing syntax, options which may appear exactly once are delimited by brackets (e.g., "[ ]"), while options that may appear multiple times are delimited by braces (e.g., "{ }"). The "|" (boolean exclusive or) symbol is used to indicate a selection from a list of choices. The delimiters are not entered into the actual descriptor file.

3. Descriptor files may contain comments. Comments are delimited by using either the standard C convention (e.g., delimited by /* */), or on a line by line basis using the # character. *C style comments may not be nested.* C style comments may not be used in `PostInstall`, `PreInstall`, or `DEINSTALL` since these are executable scripts.

4. Blank lines may be used freely and are ignored unless they are within a block of text for insertion, replacement, etc. Blank lines are ignored when searching for a block to delete or replace. Similarly, blanks, tabs, and other whitespace are ignored unless they are part of a block to insert or replace.

5. When a block of text is required, such as in adding a block of text to a community file, the characters "{" and '"}" are used as block delimiters.

6. Keywords inside a descriptor file are always prefixed with the "$" character.

7. C style `#ifdef`, `#else`, `#elif`, `#endif`, and `#ifndef` constructs may be used in descriptor files, along with the standard C boolean operators. These constructs may not span segment descriptor sections. The constants which may be used in these constructs are defined in subsection 5.3.

8. During installation, the COE installation software sets four environment variables: `INSTALL_DIR` is the absolute pathname for where the segment will be loaded (`PreInstall`) or was loaded (`PostInstall`). `MACHINE`, `MACHINE_CPU`, and `MACHINE_OS` are set to describe the type

of platform on which the software has been loaded. Valid values for these environment variables are listed in subsection 5.3.

9. Parameters which follow a keyword are given on the same line as the keyword and are separated by colons. The exception to this rule is when the keyword signals the beginning of a variable length list. For example,

   ```
   $PATH:/etc
   ```

   specifies a pathname while

   ```
   $LIST
   f1
   f2
   f3
   ```

   specifies a list of files.

10. Some segment descriptors, such as the `Requires` descriptor, specify the name of another segment that the COE installation tools must search for at install time. To speed up the search process, segment names are expressed in the form

    ```
    segment name:prefix:home dir:[version:{patches}]
    ```

    where *segment name* is the name of the segment, *prefix* is the segment's prefix, *home dir* is the segment's expected home directory, while *version* and *patches* are optional. home dir is searched first, and if the segment name found there is the same as that specified, a match is declared successful. If home dir does not exist, is not a segment, or the segment name does not match, an exhaustive search is performed on all segments on all mounted disk partitions.

11. Some segment descriptors allow a version number or patch level to be specified. See the previous `Requires` example. If no version number is specified, any version found is successful. If a version number is specified, an ordinary lexical comparison of primary version numbers is made with zeroes inserted for any missing digits. For example, a version number such as 3.4/SunOS-4.1.3 is truncated to just the primary version number which is then expanded to be 3.4.0.0 for comparison purposes.

12. Some descriptor file features require prior DISA Chief Engineering approval, or are restricted to COE component segments. These are described in the sections which follow.

COE compliance requires the following:

¥ Segments shall include all required files shown in Table 5-1.

¥ Segments shall not delete the segment directory during a `DEINSTALL` script.

¥ Segments shall fully specify all dependencies and conflicts through the `Requires` and `Conflicts` descriptor files. (Circular dependencies are not allowed.)

¥ Segments shall fully specify disk and memory requirements (memory may be omitted for data segments) in the `Hardware` file.

¥ Segments shall not use `PostInstall`, `PreInstall`, or `DEINSTALL` to make modifications that the COE installation software will make.

¥ Segments shall use the `ReleaseNotes` file to convey information meaningful to an operator, not the system integrator. `ReleaseNotes` files shall not include company names, names of individuals, nor DISA software trouble report numbers.

¥ Segments shall specify a version number and date in the `VERSION` descriptor file, and shall increment the version number for each subsequent release. Version numbers shall fully comply with the requirements stipulated in subsection 3.1 of this document.

¥ Segments shall use individual segment descriptor files or a single descriptor file, `SegInfo`, but not both. Full COE compliance requires using the `SegInfo` file since segment descriptor files will be phased out in a subsequent release.

5.5.1 AcctGroup

Syntax for the `AcctGroup` descriptor is

```
group name:group ID:shell:profile flag:home dir:default profile name
```

where

group name is an alphanumeric string used to identify this account group. The account group name must be unique (i.e., no other account group may have the same name).

group id is a Unix group id to be inserted into the password file for accounts created from this group. The user id is calculated automatically by examining the password file for user accounts within the same group and then adding 1 to the highest user id. Group ids less than 100 should be avoided.

shell is the Unix shell to execute when logging in (e.g., `/bin/csh`, `/bin/sh`)

profile flag is 0 if no profiles are allowed, otherwise 1.

home dir is the home directory for the given account group (e.g., `/h/AcctGrps/SecAdm`).

default profile name is an alphanumeric string identifying the account group's default profile. This name is ignored unless the profile flag is nonzero.

In effect, `AcctGroup` is a template of what to enter into the `/etc/passwd` file for accounts within this group.

Group names and profile names are not case sensitive. The maximum number of characters in a group name, including embedded blanks, is 15. The maximum number of characters in a profile name is 15. The maximum number of characters in the home directory pathname is 256.

If the account group is to have a default profile, the installation software will automatically create the profile with the name specified. The profile will be set up to have a classification level of TOP SECRET (unless the segment specifies otherwise), all possible object permissions enabled (see the `Permissions` descriptor), and all possible menu and icon entries enabled.

The profile classification can be explicitly stated by including a line of the form

```
$CLASSIF:classification
```

in the descriptor file. Valid classification values are

UNCLASS
CONFIDENTIAL
SECRET
TOP SECRET

5.5.2 COEServices

Segments frequently require changes to services provided by the operating system. Make such requests through the `COEServices` descriptor to ensure proper coordination with other segments. One or more entries may follow each keyword below.

$GROUPS

Segments may add entries to the `/etc/group` file as follows:

```
$GROUPS
name:group id
```

where *name* and *group id* have the meaning defined by the Unix `group` file. If the specified name already exists in the group file but with a different group id, an error will be generated.

$PASSWORDS

Segments may occasionally need to add entries into the Unix password file to establish file ownership. The syntax is:

```
login name:user id:group id:comment:home dir:shell
```

where these entries correspond to the entries in the Unix `passwd` file. Multiple lines may be included to add multiple password entries.

The installation software inserts an "*" for the password field to ensure that these are system accounts, not actual login accounts. Segments that need to add a user account must be approved in advance by the DISA Chief Engineer, and then will generally be approved only for COE component segments.

The installation software processes the `$PASSWORDS` keyword *before* the segment is actually loaded onto disk so that `PostInstall` scripts which need to set file ownership will work properly.

$SERVICES

Ports are added to the `/etc/services` system file through the `$SERVICES` keyword. The syntax is:

```
$SERVICES[:comment]
name:port:protocol{:alias}
```

where

name is the name of the socket to add, *port* is the port number requested, and

protocol is either `tcp` or `udp`.

The optional *comment*, if provided, will be inserted into the `/etc/services` file by the installation software.

If the port number requested is already in use under another name, an error will be generated. Note that port numbers in the range 2000-2999 are reserved for COE segments.

5.5.3 Community

Many of the descriptor files direct the installation software to insert, delete, replace or otherwise alter blocks of text in ASCII files. The `Community` descriptor file is provided to issue similar commands to the installation software for which no corresponding descriptor file exists. It is intended to be a "catch all" and should be used carefully, and only when there is no other way to accomplish the modifications required. `VerifySeg` will fail any segment which attempts to use a `Community` descriptor file to modify a file that is already handled by another descriptor file.

Segment developers shall use the `Comm.deinstall` descriptor file to undo changes made by the `Community` file. `Comm.deinstall` is invoked when a segment is removed, and is essentially a reverse image of the `Community` file.

The commands listed below are available for both the `Community` and `Comm.deinstall` files. Blocks of text are delimited by braces, where the opening and closing brace are on a line by themselves. When commands require that a textual search be done, embedded spaces and control characters are ignored during the search.

To illustrate how the commands work, assume the file `IDE.TEST` contains the following text:

```
# Sample file

# Define runtime vars
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data

# set a test var
setenv testvar $HOME

set filec
```

```
setenv testvar2 $HOME/data

# end of example file
```

$APPEND

Append the block of text which follows to the end of the file.

Example:

```
$APPEND
{
# This is an example to append at the end of a file
source my_script
#
}
```

$COMMENT:char

Using the character specified, find the block of text which follows and comment it out. This effectively deletes text, but has the advantage that it can easily be uncommented.

The command sequence

```
$COMMENT:#
{
# set a test var
setenv testvar $HOME
set filec
}
```

will replace the text to modify the file as follows:

```
# Sample file

# Define runtime vars
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data

## set a test var
#setenv testvar $HOME
#
#set filec

setenv testvar2 $HOME/data

# end of example file
```

Notice that the blank line between `setenv` and `set` is ignored in searching for the lines to delete, but is preserved in the commented out version of the file.

Note: Be careful to note that the '#' character is not a valid comment delimiter for all community files! (e.g., X and Motif resource files use '!' as a comment delimiter.)

$DELETE [ALL]

Find the block of text which follows and delete it from the file. If *ALL* is specified, delete every occurrence in the file.

The command sequence

```
$DELETE
{
# set a test var
setenv testvar $HOME
set filec
}
```

will delete the block of text to modify the file as follows:

```
# Sample file

# Define runtime vars
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data


setenv testvar2 $HOME/data

# end of example file
```

Notice that the blank line between `setenv` and `set` is ignored in searching for the lines to delete, but is deleted in the resulting version of the file.

$FILE filename

Name the file to which the commands that follow apply.

Example:

```
$FILE:/h/IDE/Scripts/IDE.JMCIS
```

$INSERT [ALL]

Find the first occurrence of the first block of text, then insert the second block of text immediately after it. If *ALL* is specified, insert the second block of text after every occurrence.

Example:

```
$INSERT
{
setenv OPT_DATA $OPT_HOME/data
}
{
setenv OPT_BIN $OPT_HOME/bin
setenv OPT_SRC $OPT_HOME/src
}
```

The resulting changes to the example file are:

```
# Sample file

# Define runtime vars
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data
setenv OPT_BIN $OPT_HOME/bin
setenv OPT_SRC $OPT_HOME/src

# set a test var
setenv testvar $HOME

set filec

setenv testvar2 $HOME/data

# end of example file
```

$REPLACE [ALL]

Replace the first occurrence of the first block of text, if found, with the second. If *ALL* is specified, replace every occurrence.

Example:

```
$REPLACE
{
setenv OPT_HOME /h/OPT
}
{
setenv OPT_HOME /home2/OPT
}
```

Embedded spaces and control characters are ignored in the search, but are preserved in the replacement. Case is preserved in the search and in the replacement.

`$SUBSTR: DELETE [ALL] | INSERT [ALL] | REPLACE [ALL]`

When performing a textual search, search for a matching substring instead. Insertions, deletions, or replacements are made as indicated.

`$UNCOMMENT:char`

Find the block of text which follows and uncomment it. The comment character is *char*, but the block of text which follows the `$UNCOMMENT` command does not contain the comment character.

Example (undo the effects of the `$COMMENT` example above):

```
   $UNCOMMENT:#
   {
   # set a test var
   setenv testvar $HOME
   set filec
   }
```

Blank lines will also be uncommented if there are any between

```
   # set a test var
```

and

```
   set filec
```

Consider a more complete example. The following will insert two new environment variables at the end of the file, replace `OPT_HOME` with `OPTION_HOME`, replace `OPT_DATA` with `OPTION_DATA`, and replace all occurrences of the substring "`stvar`" with "`st_var`". This example also shows the use of comments.

```
/* This is a multi-line comment
   just like in standard C.
*/
# This is a single line comment

# Assume file is in IDE Scripts subdirectory
$FILE:/h/IDE/Scripts/IDE.TEST

$REPLACE
{
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data
}
{
setenv OPTION_HOME /h/OPTION
setenv OPTION_DATA $OPTION_HOME/data
}

$SUBSTR:REPLACE ALL
{
stvar
}
{
st_var
}

$APPEND
{
#------------------------
# BEGIN xxx modifications
#------------------------

setenv my_var /h/IDE

#------------------------
# END xxx modifications
#------------------------
}
```

The resulting file `IDE.TEST` is

```
# Sample file

# Define runtime vars
setenv OPTION_HOME /h/OPTION
setenv OPTION_DATA $OPTION_HOME/data

# set a test var
setenv test_var $HOME

set filec

setenv test_var2 $HOME/data

# end of example file
#-----------------------
# BEGIN xxx modifications
#-----------------------

setenv my_var /h/IDE

#-----------------------
# END xxx modifications
#-----------------------
```

This example shows the use of comments to enclose modifications between a BEGIN/END pair. This technique is recommended when making modifications to community files to make it easier to determine changes made as segments are installed. (This technique is used by the installation software as environment extension files are modified. Therefore, developers must *not* put such comments in environment extension files.)

5.5.4 Comm.deinstall

`Comm.deinstall` is the inverse of `Community`. Its purpose is to undo modifications made to community files when a segment is removed from the system.

The corresponding `Comm.deinstall` file to undo the changes made in the example from the `Community` subsection is:

```
$FILE:/h/IDE/Scripts/IDE.TEST
$REPLACE
{
setenv OPTION_HOME /h/OPTION
setenv OPTION_DATA $OPTION_HOME/data
}
{
setenv OPT_HOME /h/OPT
setenv OPT_DATA $OPT_HOME/data
}

$SUBSTR:REPLACE ALL
{
st_var
}
{
stvar
}

$DELETE
{
#-----------------------
# BEGIN xxx modifications
#-----------------------

setenv my_var /h/IDE

#-----------------------
# END xxx modifications
#-----------------------
}
```

5.5.5 Compat

Subsequent releases of a segment are not always backwards compatible. The `Compat` descriptor is used to indicate the degree to which backward compatibility is preserved with the newly released segment. This is achieved by listing version numbers for previous releases which the current release supports. In the sense used here, backwards compatibility means that the segment being released will work with other segments that have been compiled and linked with an earlier release version.

The format of the `Compat` descriptor is a single line containing one of three possible entries:

+ALL	This indicates that the current release is backwards compatible with all previous releases.

-NONE	This indicates that the current release is not backwards compatible with any previous release.
version list	This indicates that the current release is backwards compatible to a list of versions. Version lists are denoted by the `$LIST`, `$EARLIEST`, and `$EXCEPTIONS` keywords.

For example, suppose the new `MySeg` release is version 3.2.5.4 and that it is compatible with all versions from 2.9.1 up to the present with the exception of versions 3.0.1.2 and the 3.1 version series. Then the `Compat` file would contain the following entries:

```
# First number listed is earliest compatible version
$EARLIEST
2.9.1
# Remaining version numbers are exceptions
$EXCEPTIONS
3.0.1.2
3.1
```

When a digit is omitted from the version number, or an asterisk is in place of the digit, there is an assumed wildcard in that digit position.

The `$LIST` keyword is used to indicate an explicit list of compatible versions. `$LIST` is mutually exclusive with the `$EARLIEST`/`$EXCEPTIONS` keyword pair. When specifying a list, a range can be indicated by the optional keyword `$TO`. Thus, the previous example could also have been done as

```
$LIST
2.9.1 $TO 3.0.1.1
3.0.1.3 $TO 3.0.9
3.2.0 $TO 3.2.5
```

In some cases, one or more patches must be applied to preserve compatibility. The patches are listed by number immediately after the version number by using a colon between patch numbers. This may be done only with the `$LIST` keyword. For example,

```
$LIST
2.9.1:P4:P5
3.0.1.1
3.0.2:P8 $TO 3.0.4:P7
```

This means that the current version is backwards compatible with

¥ 2.9.1, but only if patches P4 and P5 have been applied

- ¥ 3.0.1.1 with no restrictions regarding patches
- ¥ 3.0.2 through 3.0.4 with the restriction that patch P8 must be applied to version 3.0.2 and patch P7 must be applied to version 3.0.4.

If no `Compat` file exists, the present version is assumed to not be backwards compatible with any previous releases. That is, `-NONE` is assumed.

5.5.6 Conflicts

Two segments may conflict with one another so that one or the other, but not both, can be installed. The `Conflicts` descriptor is used to specify such inter-segment conflicts. The format is a list of conflicting segments in the form:

```
segment name:prefix:home dir[:version{:patch}]
```

where *segment name* is the name of the conflicting segment as given in the segment's `SegName` descriptor file, *prefix* is the conflicting segment's segment prefix, and *home dir* is the conflicting segment's home directory.

The `Conflicts` descriptor is essentially the mirror image of the `Requires` descriptor file.

5.5.7 Data

The `Data` descriptor is used to describe where data files are to be logically loaded and their scope (global, local, or segment). Only one of the three scopes may be specified in the descriptor file; that is, a data segment has one and only one scope.

The syntax is

```
$SEGMENT:segname:prefix:home dir
```

for segment data, or

```
$LOCAL:segname:prefix:home dir
```

for local data, or

```
$GLOBAL:segname:prefix:home dir
```

for global data, where *segname*, *prefix*, and *home dir* refer to the affected segment. The segname and prefix must match the name given in the affected segment's `SegName` descriptor. Figure 5-7 shows that the data to install is underneath the segment's `data` subdirectory.

Obsolete: The format here differs from earlier releases. The size required is now specified in the `Hardware` descriptor instead of the `Data` descriptor. The old format will be supported to provide backwards compatibility, but it is being phased out. Full COE compliance requires uses of the format stated here.

5.5.8 Database

This segment descriptor is not currently implemented. It is reserved for future use to implement database specific requirements as they evolve.

5.5.9 DEINSTALL

The `DEINSTALL` descriptor file is an executable, either a script or a C program, that is invoked by the installation software when the operator has elected to remove a segment. This may occur by explicitly selecting a segment to remove, or when installation of a new version of the segment is done.

If this file does not exist, the segment is assumed to be permanent and can not be removed except when installing a new version. If a new version is installed and this file does not exist, the installation software will use the information in the descriptor directory to undo changes made by the previous installation of the segment and then simply delete the directory.

For security reasons, the `DEINSTALL` script is not run with root level privileges, unless the `$ROOT` keyword is given in the `Direct` descriptor, described below.

Obsolete: Previous versions of the COE allowed `DEINSTALL` to run with root privileges. This capability is no longer the default. The `$ROOT` keyword *must* be used instead.

5.5.10 Direct

The segment descriptor `Direct` allows a segment to issues special instructions to the installation tools. If the segment is part of an aggregate, the directives below apply *only* to the segment in whose `SegDescrip` subdirectory the directives appear.

`$ACCTADD: executable`

This keyword informs the installation software that the specified *executable*, in the segment's `bin` subdirectory, should be run each time a user account is added to the system. `VerifySeg` will flag use of this keyword as a warning to highlight that it is being used. Prior permission must be given by the DISA Chief Engineer before this keyword can be used.

`$ACCTDEL: executable`

This keyword informs the installation software that the specified *executable*, in the segment's `bin` subdirectory, should be run each time a user account is deleted from the system. `VerifySeg` will flag use of this keyword as a warning to highlight that it is being used. For security reasons, prior permission must be given by the DISA Chief Engineer before this keyword can be used.

`$NOCOMPRESS`

The `MakeInstall` tool automatically compresses segments to reduce the amount of space required on disk or tape, and to reduce the download time. The installation tools automatically decompress segments at installation time. The `$NOCOMPRESS` keyword indicates that compression is *not* to be performed.

`$REBOOT`

The presence of this keyword indicates that the installation software should automatically reboot the computer after the segment is loaded. If several segments have been selected for loading at one time, the reboot operation will not occur until all segments have been processed. The operator will be notified before the reboot occurs and given the option to override the reboot directive.

`$REMOTE[:XTERM | :CHARBIF]`

This keyword indicates that the functions (*all* functions) provided by this segment can be executed remotely. At installation time, the installation software will note that this segment can be executed remotely. If the `XTERM` attribute is present, it indicates that the segment can also be accessed via an "xterm" capability, and output will be routed to the display surface pointed to by the

DISPLAY environment variable setting. If the CHARBIF attribute is present, it indicates that the segment supports a character based interface. CHARBIF and XTERM will normally be mutually exclusive.

By default, segments are assumed to be locally executable only.

$ROOT:PostInstall | PreInstall | DEINSTALL

The presence of this keyword indicates that the specified descriptor must be run with root privileges. A separate $ROOT entry is required for each descriptor. VerifySeg will flag use of this keyword as a warning to highlight that it is being used. For security reasons, prior permission must be given by the DISA Chief Engineer before this keyword can be used.

5.5.11 FileAttribs

The FileAttribs descriptor allows a segment to specify the attributes (owner, read/write permissions, group) for each file in the segment. It is created by the tool MakeAttribs (see Appendix C). The installation tools, just prior to PostInstall, will use information in this file to set file attributes.

FileAttribs has certain restrictions due to security and segment integrity considerations. The following will be ignored:

¥ Files within the SegDescrip subdirectory
¥ Files outside the segment
¥ Requests to set root ownership
¥ Requests to set Unix "sticky bits" (e.g., chmod 4644)

If FileAttribs is not provided by the segment, the installation tools will automatically do the following for all except COTS segment types:

¥ chmod 554 for all files in the bin subdirectory
¥ chmod 664 for all files in the data subdirectory
¥ for account groups, set owner to the same group id as specified in the AcctGrps descriptor for all subdirectories except SegDescrip
¥ for other segment types, set owner to the same group id as the affected segment for all subdirectories except SegDescrip.

5.5.12 FilesList

FilesList is a list of files and directories that make up the current segment. It is required for COTS segments. For other segment types, it is useful for documenting community files modified or used by the segment.

`FilesList` may contain the following keywords:

`$DIRS`	a list of directories which this segment adds to the system. All files in the directory are assumed to belong to the segment.
`$FILES`	a list of files which this segment adds to the system.
`$PATH`	a shortcut for specifying a pathname. Succeeding `$DIRS` or `$FILES` are relative with respect to the path specified.

A keyword must precede any list so that it will be clear whether a directory or a file is intended.

As an example, assume a segment to be installed creates the following four subdirectories

```
/h/data/test/data1
/h/data/test/data2
/h/data/opt/data3
/usr/opt/temp
```

and adds three files (`f1`, `f2`, `f3`) to the `/etc` subdirectory. Then the file `FilesList` could contain the following entries:

```
$PATH:/h/data
$DIRS
test/data1
test/data2
opt/data3
$DIRS
/usr/opt/temp
$PATH:/etc
$FILES
f1
f2
f3
```

The `$DIRS` keyword before `/usr/opt/temp` is not necessary, but is shown to illustrate that `FilesList` may contain multiple occurrences of the keywords.

5.5.13 Hardware

The `Hardware` descriptor defines the computing resources required by the segment. Keywords `$CPU` and `$MEMORY` may appear only once; both are required for all segments, except that `$MEMORY` may be omitted for a data segment. `$DISK` and `$PARTITION` are mutually exclusive, but one must appear in the segment descriptor. `$DISK` may appear only once, but `$PARTITION` may appear multiple times. `$OPSYS` and `$TEMPSPACE` are optional.

`$CPU:platform | ALL`

platform is one of the supported platform constants listed in subsection 5.3 for `MACHINE` or `MACHINE_CPU`, or *ALL*. If ALL is given, it indicates that the segment is hardware independent (e.g., a data segment or a segment comprised of shell scripts). If platform is a generic constant (e.g., HP or PC), it applies to all platforms of that class. Thus,

```
    $CPU:PC
```

indicates that the software can be loaded on any PC, whether the PC is a 386, 486, or Pentium class machine. However,

```
    $CPU:PC386
```

indicates that the software can be loaded on a 386 or better class platform. Similarly, HP712 indicates that the software can be loaded on an HP712 or better class platform that is binary compatible with the HP712.

`$DISK:size[:reserve]`

size is expressed in kilobytes and is the size of the segment, including all of its subdirectories, at install time. The COE tool `CalcSpace` (see Appendix C) will compute the disk space occupied by a segment and update this keyword. *reserve* is also expressed in kilobytes and is the *additional* amount of disk storage to reserve for future segment growth.

`$MEMORY:size`

size is expressed in kilobytes of RAM required.

`$OPSYS:operating system | ALL`

operating system is one of the supported platform constants listed in subsection 5.3 for `MACHINE_OS`, or *ALL*. If ALL is given, it indicates that the segment is operating system independent. Dependencies on a particular version of the operating system are defined in the `Requires` descriptor where a dependency on a specific segment (e.g., operating system with a particular version) is described.

$PARTITION:diskname:size[:reserve]

This keyword allows segments to reserve space on multiple disk partitions. The installation software will not split a segment across disk partitions, but the segment may do so in a `PostInstall` script. Use of multiple disk partitions is discouraged.

size and *reserve* have the same meanings as for `$DISK`. *diskname* is either an explicit partition name (e.g., `/home2`) or an environment variable name of the form `DISK1`, `DISK2`, ... `DISK99`. The installation software will set the environment variables `DISK1`, `DISK2`, etc. to the absolute pathname for where space has been allocated. These environment variables are defined for `PreInstall` and `PostInstall`, but not for `DEINSTALL`. `$PARTITION` keywords are assumed to be in sequential order, so that environment variable `DISK1` will refer to the first keyword encountered, `DISK2` to the second, etc.

For example, suppose a TDA is compiled to run on both and HP and a Solaris workstation. Assume for the HP it requires 512 K of memory, requires 1 MB of disk storage for the program and its data files, and will expand over time to a maximum of 4 MB. For Solaris, assume it requires 576 K of memory, 1.5 MB for initial disk space, and will expand to 5 MB. For a PC, assume the requirements are the same as for the Solaris machine. The proper `Hardware` file is

```
#ifdef HP
    $CPU:HP
    $DISK:1024:3072
    $MEMORY:512
#elif SOL
    $CPU:SOL
    $DISK:1536:3584
    $MEMORY:576
#elif PC && NT
    $CPU:PC486
    $DISK:1536:3584
    $OPSYS:NT
    $MEMORY:576
#endif
```

Note that this example indicates that the information described is the same for all HP platforms, the same for all Solaris platforms, but that it only applies to PC486 or better machines running Windows NT.

As another example, assume a data segment is to be allocated across three disk partitions. Further assume that the first partition must be `/home5` and requires 10 MB, but the remaining space required is 20 MB each and can be on any available disk partition. The proper `$PARTITION` statements are:

```
$PARTITION:/home5:10240
$PARTITION:DISK2:20480
$PARTITION:DISK3:20480
```

Assume that the installation software is able to allocate space on `/home5` as requested, and that the remainder of the space requested is on `/home18`. The installation software will set the following environment variables:

```
setenv DISK1      /home5
setenv DISK2      /home18
setenv DISK3      /home18
```

`$TEMPSPACE:size`

Some segments may need temporary space during the installation process. The `$TEMPSPACE` keyword requests that *size* kilobytes of disk space be allocated for temporary use during the installation process. If space is available, the installation software sets the environment variable `COE_TMPSPACE` to the absolute path where space was allocated. If space is not available, an error message is displayed to the operator and the segment installation fails. The installation software automatically deletes the allocated space when segment installation is completed. Space is allocated prior to executing `PreInstall`.

5.5.14 Icons

The `Icons` descriptor is used to define the icons that are to be made available on the desktop to launch segment functions. The format of the descriptor is a list of files underneath `data/Icons` that define icon bitmaps and their associated executables. Refer to the Executive Manager API documentation for a description of the file format.

5.5.15 Installed

The installation software creates the file `Installed` as segments are loaded. The file specifies the segment that was loaded, the date and time of the installation, which workstation was used to do the installation, and the version number of the software used to do the installation. This file is located underneath the segment descriptor directory.

5.5.16 Menus

Use the file `Menus` to list the names of menu files contained within the segment. Figure 5-2 shows that segment menu files are located underneath `data/Menus`. Refer to the Executive Manager API documentation for the menu file format.

For account groups, this descriptor is simply a list of the account group's menu files. For other segments, the format of each line is

```
menu file[:affected menu file]
```

where *menu file* is the name of a menu file underneath the segment's `data/Menus` subdirectory, and *affected menu file* is the account group menu file to update. If multiple account groups are affected, as listed in the `SegName` descriptor, each affected account group is updated. If no affected menu file is listed, then menu file is simply added to the list of menu files which comprise the account group's menu templates.

For example, suppose a segment called `ASWTDA` has four menu files in the `data/Menus` subdirectory: `System`, `MoreStuff`, `ASWTDA`, and `Logging`. Assume that `System` is to be added to the affected account group's `System` menu file, and `MoreStuff` is to be added to the affected account group's `Default` menu file. The proper entries are as follows:

```
System:System
MoreStuff:Default
ASWTDA
Logging
```

Obsolete: Earlier releases of the COE provided a `$PATH` keyword to indicate where menu files were located relative to the segment's home directory. This method is obsolete and will be phased out. Full COE compliance requires adherence to Figure 5-2.

5.5.17 ModName

Obsolete: `ModName` has been replaced by the `SegName` descriptor file. `ModName` is supported for backwards compatibility, but will be phased out in a later release. Full COE compliance requires use of `SegName` instead of `ModName`.

5.5.18 ModVerify

Obsolete: `ModVerify` has been replaced by `SegChecksum`.

5.5.19 Network

The `Network` descriptor is used to describe network related parameters. Use of this descriptor requires prior approval by the DISA Chief Engineer, and its use is restricted to COE component segments. `VerifySeg` will strictly fail any segment that includes this descriptor unless it is a COE component segment.

One or more entries may follow each keyword listed below.

$HOSTS

IP addresses and hostnames are generally established by a system or network administrator. Segments may add IP addresses and host names as follows:

```
$HOSTS
LOCAL | REMOTE :IP address:name{:alias}
```

where *IP address, name*, and *alias* are as defined for the Unix `/etc/hosts` file. If the IP address specified already exists in the network hosts file, the name and alias entries are added as alias names. If LOCAL is specified, the entry is made only to the local network hosts file. If REMOTE is specified, the entry is applied to the NIS+ or DNS server. If REMOTE is specified but neither NIS+ or DNS are installed, it will default to LOCAL.

Segments should rarely need to directly add host table entries. `VerifySeg` will issue a warning for any segment which adds host table entries.

$MOUNT

The `$MOUNT` keyword is used to specify NFS mount points. The syntax is

```
hostname:NFS mount point:target dir
```

where *hostname* is the name of a workstation on the network, *NFS mount point* is the file partition to mount, and *target dir* is where to mount the requested partition on the local machine. If target dir does not exist on the local machine, it will be created.

For example, the sequence

```
$MOUNT
dbserver:/home3/USERS:/h/USERS
```

will perform the Unix equivalent of

```
mount dbserver:/home3/USERS /h/USERS
```

If the hostname specified is the same as the local machine, a mount is not performed. Instead, the NFS mount point is made available for other workstations to mount. If a mount is performed as a result of processing this keyword, the system will automatically reboot the system after segment installation is completed. It performs as if the `$REBOOT` keyword (see the `Direct` descriptor) was encountered; that is, the operator is notified that a reboot is required and given an option to override the reboot directive.

$NETMASK:mask

This keyword allows a COE component segment to set the subnet mask to *mask*. This should rarely be required since the netmask is normally established as part of the kernel COE. If two COE component segments attempt to set the netmask, the last segment loaded succeeds.

$SERVERS

Most COE services are implemented as servers. This keyword allows a segment to list the servers, by symbolic name, that it contains. These servers are registered with the COE so that other segments can find their location through the `COEFindServer` function (see Appendix C).

Each name listed is added to a table maintained by the COE of all servers in the system. This table is used by the System Administration software to allow a site administrator to indicate which platform actually contains the server. The name given is added as an alias to the network host table for the workstation that contains the server. If NIS+/DNS are being used, the alias is added to the NIS+/DNS managed host table. Otherwise, it is added to `/etc/hosts`.

For example, assume a COE component segment contains two servers named `masterTrk` and `masterComms`. Assume that this segment is loaded on two workstatations: `sys1` and `garland`. Some servers are designed to recognize whether they are the master server, or are a slave to a master server located elsewhere. For this reason, the COE must handle situations where the same segment is loaded on a server and a client machine. Assume in this example that the segment operators as a master server on `sys1`, but as a slave on `garland`.

The following statements identify the servers contained within this segment:

```
$SERVERS
masterTrk
masterComms
```

When the segment is loaded, the installation software performs the following actions:

1. Add `masterTrk` and `masterComms` to the COE maintained list of servers if they are not already there.
2. Check to see if `masterTrk` or `masterComms` already exist in the network host table. If so, processing is completed.
3. Otherwise, ask the operator if this is the server platform for `masterTrk` and `masterComms`.
4. If the operator answers "no" to the previous question, processing is complete.
5. If the answer is "yes," update the network host table to contain `masterTrk` and `masterComms` as aliases for the machine being loaded.

Note that this approach does not require the server (`sys1`) to be loaded prior to the client (`garland`). Furthermore, the site administrator can later change the configuration because all necessary information is available to the System Administrator software. Also note that the segment does not require the actual hostnames or IP addresses.

Hostnames are site specific and can not be predicted in advance. Therefore, the COE requires that segments use meaningful symbolic names as illustrated here instead of making assumptions about a specific hostname, or naming convention.

5.5.20 Permissions

The Security Administrator software provides the ability to describe objects (files, data fields, executables, etc.) which are to be protected from general access. This information is used to create profiles which limit an operator's ability to read or modify files. Applications may query the security software to determine the access permissions granted to the current user. The `Permissions` file is the mechanism for segments to describe objects and permissions to grant or deny for the objects.

This descriptor is a sequence of lines of the form:

```
   object name:permission abbreviation:permission
```

object name is the item to be controlled, *permission* is the type of access to grant or deny (add, delete, read, etc.), and *permission abbreviation* is a single character abbreviation for the permission.

Permission abbreviations specified for an account group must agree with all segments which become part of the group. The following are reserved abbreviations and their meaning:

A - Add
D - Delete
E - Edit
P - Print
R - Read
V - View
X - Transmit

Segments may use additional abbreviations as required.

For example, suppose a segment generates reports that are to be protected. Permissions to grant or deny for reports are delete, print, read, or archive. The proper `Permissions` file is (Z is used to indicate archive permission in this example):

```
Reports:D:Delete:P:Print:R:Read:Z:Archive
```

If the `Permissions` file is missing, the security software will report that for the requested object, no access permissions are to be granted.

5.5.21 PostInstall

Most of the work required to install segments is performed by the COE installation software through information contained in the descriptor directory. However, additional segment dependent steps must sometimes be performed. `PostInstall` is an executable, either a script or a compiled program, that segment developers may provide to handle segment specific installation functions *after* the segment has been copied to disk and installed by the COE. During installation, `PostInstall` may invoke functions (e.g., prompt the user) described in Appendix C.

The `PostInstall` descriptor must *not* do any operations that are performed by the COE installation software. For security reasons, the `PostInstall` script is not run with root level privileges, unless the `$ROOT` keyword is given in the `Direct` descriptor.

Obsolete: Previous versions of the COE allowed `PostInstall` to run with root privileges. This capability is no longer the default. The `$ROOT` keyword *must* be used instead.

5.5.22 PreInstall

The `PreInstall` descriptor file is identical to `PostInstall` except that it is invoked by the installation software *before* the segment is loaded onto the disk. It must *not* do any operations that are performed by the COE installation software. For security reasons, the `PreInstall` script is not run with root level privileges, unless the `$ROOT` keyword is given in the `Direct` descriptor.

Obsolete: Previous versions of the COE allowed `PreInstall` to run with root privileges. This capability is no longer the default. The `$ROOT` keyword *must* be used instead.

5.5.23 PreMakeInst

`PreMakeInst` is an optional executable program or script that is invoked by the `MakeInstall` tool. It's purpose is to allow a segment to perform "cleanup" operations, such as deleting temporary files, before `MakeInstall` writes the segment to the distribution media. `MakeInstall` sets the environment variables `INSTALL_DIR`, `MACHINE`, `MACHINE_CPU`, and `MACHINE_OS` prior to invoking `PreMakeInst`.

5.5.24 Processes

Use the `Processes` descriptor to identify background processes (see subsection 5.7 below). The format of the descriptor is a keyword which identifies the type of process, followed by a list of processes to launch in the form

```
process {parameters}
```

where *process* is the name of the executable to launch, and *parameters* are optional process dependent parameters. Output from the process is piped to `/dev/null`. For example, suppose `TestProc` is a background process which accepts two parameters, `-t` and `-c`. It will be launched in a manner equivalent to

```
TestProc -t -c >& /dev/null &
```

Valid keywords to identify process type are:

$BOOT	specify a list of processes to launch at boot time
$BACKGROUND	specify a list of background processes
$SESSION	specify a list of login session processes
$SESSION_EXIT	specify a list of processes to run prior to terminating a login session

Executables are assumed to be in the segment's `bin` subdirectory. The `$PATH` keyword can be used to indicate a different location. The syntax for the `$PATH` keyword is

```
$PATH:pathname
```

where *pathname* may be either a relative or an absolute pathname. If the pathname is relative, it is relative to the segment's home directory.

5.5.25 ReleaseNotes

Use the ASCII file `ReleaseNotes` to provide information useful to an operator in understanding the new functionality being provided by the segment, or the problems being fixed. It is *not* a help file, nor is it information targeted to the system integrator. Therefore, it must not reference problem report numbers, version numbers, release dates, individuals or companies, point of contact, or similar information. (This type of information is contained elsewhere, such as in the `VERSION` file, and duplication of information may lead to conflicting or confusing information for the operator.) The `ReleaseNotes` file must not contain any tabs or embedded control characters.

An example of a "poor" `ReleaseNotes` file is

```
Release: 5.6.3
Point of Contact: John Doe, Tritron Company
Phone: (619) 555-1234

1. Implemented NCR #302
2. Added check for memory overflow
3. Fixed problem with double scrolling in STR #307
```

An example of a "good" `ReleaseNotes` file is

```
This release fixes two known problems:

(a) Calculation of range and bearing for polar latitudes
has been corrected

(b) Display of garbled latitude/longitude in the Track Summary
display for ownship has been corrected

The following new features are added with this release

1. Search and Rescue TDA added.

2. Option added to restrict operator deletion of comms
messages to only those created by the operator.
```

5.5.26 ReqrdScripts

Use the `ReqrdScripts` descriptor to define the files that establish the runtime environment (account group segment types), or to define files to extend the runtime environment (all other segment types). For account group segments, the syntax is one or more lines of the form:

```
script name:C | L
```

where *C* means to copy and *L* means to create a symbolic link. This flag is used when login accounts are created to either copy environment files to the user's login directory, or to create a symbolic link. There can be a maximum of 32 scripts. A script name is restricted to a maximum length of 32 characters.

For example, the `ReqrdScripts` file for the System Administrator account group is

```
.Xdefaults:C
.c_p:C
.cshrc:C
.login:C
.mwmrc:C
.xsession:C
```

The descriptor format for segment types other than account group is slightly different:

```
script name:env ext name
```

where *script name* is the name of a script in the affected account group's `Scripts` subdirectory, and *env ext name* is the name of an environment extension file in the present segment's `Scripts` subdirectory.

For example, assume a segment loaded under `/h/TstSeg` with a segment prefix `TST` is to be added to the System Administrator application, and it requires extending the `.cshrc` file. The proper `ReqrdScripts` entry is:

```
.cshrc:.cshrc.TST
```

The installation tools will insert the statements

```
if (-e /h/TstSeg/Scripts/.cshrc.TST) then
  source /h/TstSeg/Scripts/.cshrc.TST
endif
```

into the file /h/AcctGrps/SysAdm/Scripts/.cshrc. When the segment TstSeg is deleted, the installation tools will remove these statements.

Obsolete: Earlier releases of the COE provided a $PATH keyword to indicate where script files were located relative to the segment's home directory. This method is obsolete and will be phased out. Full COE compliance requires adherence to Figure 5-2.

5.5.27 Requires

Segment dependencies are stated through the Requires descriptor. The format is:

```
[$HOME_DIR:pathname]
segment name:prefix:home dir:[version{:patch}]
```

Segments will not be loaded until all segments they depend upon are loaded. For this reason, the parent segment for an aggregate must *not* list child segments in the Requires descriptor.

The optional $HOME_DIR keyword is used in situations where a segment must be loaded onto the disk in a particular place. This technique should be avoided.

For example, assume the segment TEST must be installed in the directory /home3/tmp/TEST, it requires version 3.0.2 of segment SegA with patches P1 and P4, and also requires SegB version 5.1.1. The Requires descriptor is

```
$HOME_DIR:/home2/tmp/TEST
SegA Name:SEGA:/h/SegA:3.0.2:P1:P4
SegB Name:SEGB:/h/SegB:4.1.1
```

In some cases, it may be possible that a segment dependency can be fulfilled by one or more segments. This is indicated by bracketing such segments with braces and using the keyword $OR between acceptable alternatives.

As an example, suppose the segment TEST above has a dependency that can be satisfied by SegA, or the combination of SegB and SegC. The proper Requires descriptor is

```
$HOME_DIR:/home2/tmp/TEST
{
  SegA Name:SEGA:/h/SegA
$OR
  SegB Name:SEGB:/h/SegB
  SegC Name:SEGC:/h/SegC
}
```

Multiple bracketed alternatives may appear in the same descriptor.

> **Obsolete:** In earlier releases, the parent segment for a child had to be listed in the `Requires` descriptor. This is no longer required because by virtue of naming the aggregate parent in `SegName`, there is an implied dependency.

5.5.28 Security

The `Security` descriptor contains a single entry indicating the highest classification level for the segment (UNCLASS, CONFIDENTIAL, SECRET, TOP SECRET). If the segment contains items with multiple classification levels, the highest classification level must be specified.

> **Note:** This file is used only to determine whether or not software should be loaded onto a workstation. It should not be confused with data labeling or other security features provided by trusted systems.

5.5.29 SegChecksum

The file `SegChecksum` is an optional file created by integration software. It contains information necessary for the System Administrator software to perform an integrity check on the installed software. If the file does not exist, the integrity check can not be performed on the segment.

5.5.30 SegInfo

`SegInfo` is an ASCII descriptor file which contains segment descriptor information in one or more sections. Table 5-2 lists the available sections. Refer to subsection 5.5 above for more information.

5.5.31 SegName

The `SegName` descriptor provides the following information:

¥ segment type (`$TYPE` keyword)
¥ segment name (`$NAME` keyword)
¥ segment prefix (`$PREFIX` keyword)
¥ segment attributes (`$TYPE` keyword)

- ¥ if applicable, affected account group, or affected segment for patches (`$SEGMENT` keyword)
- ¥ if applicable, name of parent or child segments (`$PARENT`, `$CHILD` keywords)

The keywords `$TYPE`, `$NAME`, and `$PREFIX` are required for each `SegName` descriptor. Additional keywords required depend upon segment type. COE component segments may not contain `$SEGMENT`, `$PARENT`, or `$CHILD` keywords. All other segments must have one `$PARENT` line, or one or more `$CHILD` lines, or one or more `$SEGMENT` lines.

$TYPE

The syntax for segment type is as follows:

```
$TYPE:segment type[:attribute]
```

where valid segment types are

```
COTS
ACCOUNT GROUP
SOFTWARE
DATA
PATCH
```

and valid segment attributes are

```
AGGREGATE
CHILD
COE CHILD
COE PARENT
```

Segment types are mutually exclusive; only one segment type may be given. Segment attributes are also mutually exclusive.

`AGGREGATE` is used to indicate that the segment being defined is the aggregate parent segment. It is valid only for account group, data, and software segment types. Aggregates must list one or more child segments with the `$CHILD` keyword. The COE does not allow an aggregate of aggregates. That is, it is not valid for Aggregate A to have a child B which is also an aggregate.

`CHILD` is used to indicate that the segment being defined is an aggregate subordinate segment. The parent segment must be listed using the `$PARENT` keyword.

`COE PARENT` is used to indicate that the segment being defined is the primary COE segment. It's home directory will be `/h/COE`.

`COE CHILD` is used to indicate that the segment being defined is a COE component segment other than the parent. The installation tools will verify that the segment is an authorized COE component and if not will reject the segment.

$NAME

The syntax for the $NAME keyword is

```
$NAME:name
```

where name is a string of up to 32 alphanumeric characters. Embedded spaces may be used for readability, but the string must not contain tabs or other control characters.

$PREFIX

The syntax for $PREFIX is

```
$PREFIX:segment prefix
```

$SEGMENT, $CHILD, $PARENT

The syntax for these three keywords is the same.

```
keyword:name:prefix:home dir
```

The descriptor file may contain one and only one $PARENT keyword. Multiple affected segments or child segments may be listed by listing each segment on a separate line.

Obsolete: For backwards compatibility, $COMPONENT is equivalent to the $CHILD keyword. $COMPONENT is obsolete and will be phased out.

5.5.32 SegType

Obsolete: SegType has been replaced by the SegName descriptor file. SegType is supported for backwards compatibility, but will be phased out in a later release. Full COE compliance requires use of SegName instead of SegType.

5.5.33 Validated

The COE requires strict adherence to integration and test procedures to ensure that a fielded system will operate correctly. To facilitate integration and testing, the VerifySeg tool creates the file Validated to confirm that a segment has been tested for COE compliance. Subsequent tools in the development, integration, and installation process use this file to determine if a segment has been altered, thus indicating that the segment needs to be revalidated.

The following information is captured:

¥ the version of `VerifySeg` used to validate the segment
¥ the date and time validation was performed
¥ who performed the validation
¥ a count of all errors and warnings produced by `VerifySeg` for the segment
¥ a checksum computed to enable detection of modifications made after the segment was validated.

5.5.34 VERSION

The format of the `VERSION` descriptor is

```
version #:date[:time]
```

where *version #* is the version number for the segment, *date* is the version date (in mm/dd/yy format), and *time* is an optional time stamp (in the format hh:mm). Version numbers must adhere to the rules defined in subsection 3.1.

5.6 Segment Installation

Segment installation requires some form of electronic media (tape, disk, etc.) that contains the segments, and that has a table of contents which lists the available segments. `MakeInstall` is the tool which creates this electronic media. However, it is important to identify the operations (e.g., compression) performed on segments, and the sequence in which these operations are performed.

Installation requires reading the table of contents created by `MakeInstall`, selecting the segments or variants to install, and then copying the segments to disk. Segments may actively participate in the installation process through `PostInstall`, `PreInstall`, and `DEINSTALL` scripts. This subsection details both the `MakeInstall` tool, and the installation sequence.

5.6.1 MakeInstall Flowchart

Figure 5-13 shows the sequence of operations performed by the `MakeInstall` tool.

1. `MakeInstall` is given a list of segments that are to be processed. For each segment in the list:
   a) If the segments is not already on disk, it is extracted from the repository and placed in a temporary location.
   b) A check is made to ensure that the segment is a valid segment.
   c) If the segment is invalid, an error message is displayed. If the segment was checked out of the repository and placed in a temporary location, the temporary segment is deleted. `MakeInstall` then terminates.

2. If all segments are valid, a worklist is created. The worklist is sorted to ensure that segments which have dependencies appear in the list *after* the segments they depend upon. This ensures that at install time, a tape will not have to be rewound because of segment dependencies.

3. For all segments in the worklist:
   a) Prepare the segment by executing the segment's `PreMakeInst` descriptor if it exists. `PreMakeInst` is prevented from modifying the segment's `SegDescrip`. Otherwise, `PreMakeInst` could invalidate the segment validation step above.
   b) Unless the segment specifies otherwise, all segment subdirectories except `SegDescrip` are compressed.
   c) The compressed segment and its descriptor directory are written out to the specified electronic media.

d) If the segment was extracted from the repository and placed in a temporary location, the temporary segment is deleted.

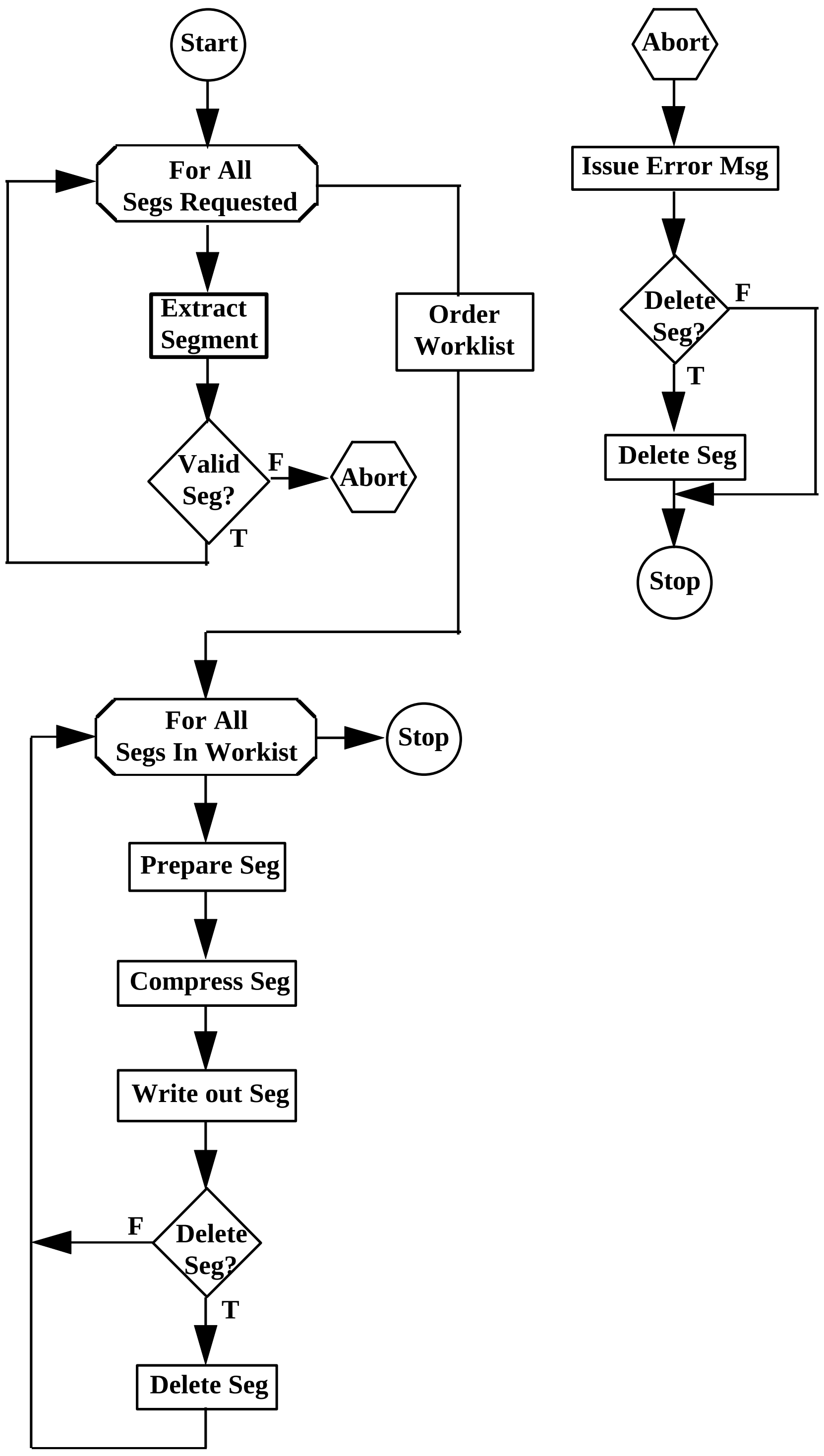

Figure 5-13: MakeInstall Flowchart

5.6.2 Installation Flowchart

Figure 5-14 is a detailed flowchart for the segment installation process. The sequence of `PreInstall`, `PostInstall`, and `DEINSTALL` executions is the most significant aspect of the flowchart. Directives contained in the `Direct` descriptor may affect the sequence (e.g., use of `$REBOOT` and `$ROOT` keywords), but such details are omitted for clarity. The installation software automatically removes patches when a segment is replaced, and deletes any temporary space (`$TEMPSPACE` keyword) allocated for the segment. These details are also omitted for clarity.

1. A load device is selected (tape, disk, etc.) and the table of contents created by `MakeInstall` is read.

2. Segments found in the table of contents which do not match the target platform are removed from consideration. Similarly, a check is made to ensure that an operator can not inadvertently load a segment for which he is not authorized. The environment variables `MACHINE`, `MACHINE_CPU`, and `MACHINE_OS` are set to indicate the hardware platform. (`MACHINE` is now obsolete.)

3. The media may have variants defined. If variants are defined:
   a) The operator may select a variant to load.
   b) If a custom installation is desired, the operator is presented with the table of contents with all segments in the selected variant highlighted. The operator may add or delete segments from this list.

   If variants are not defined, the operator is shown the table of contents and must manually select the desired segments.

4. For all segments selected, a check is made to see if the segment is loadable. To be loadable, all dependent segments must either be selected or already on disk. Conflicting segments must not be selected, nor can they already be loaded on disk.

5. For all segments selected:
   a) The installation tools determine where to load the segment. The environment variable `INSTALL_DIR` is set to the absolute pathname for where the segment will be loaded. Segments can *not* assume that any environment variables other than `MACHINE`, `MACHINE_CPU`, `MACHINE_OS`, and `INSTALL_DIR` are defined, or those set to refer to disk space (`COE_TMPSPACE`, `DISK1`, etc.).
   b) If an old version of the segment already exists on disk, the old segment's `DEINSTALL` script is run.

c) The new segment's `PreInstall` script is loaded and executed. Note that the new segment is *not* yet on disk.
d) The old segment is deinstalled by the installation tools. Modifications made through the descriptor files are reversed.
e) The old segment is deleted from disk.
f) The new segment is loaded from tape onto disk and decompressed.
g) The installation tools process commands from the new segment's descriptor files.
h) The new segment's `PostInstall` script is run. `PostInstall` may invoke load time tools described in Appendix C (e.g., to prompt the user).
i) A status message is displayed indicating whether or not the segment was successfully installed.

6. If any of the segments installed requested a reboot, the operator is notified and asked for confirmation. If the operator confirms, the system is rebooted.

5.6.3 Database Creation

The DBMS should be operating in its maintenance mode (e.g. Oracle's command STARTUP DBA EXCLUSIVE) when an application server segment is installed. This prevents users from accessing data objects during their creation and possibly corrupting either the segment or the database instance.

Table 5-3 shows, in broad outline, the sequence of steps performed by an application's server segment when it is creating the database. The first three steps must be performed by a database account with DBA privileges. The owner account (and there may be more than one) should be restricted so it can only create objects in the tablespaces designated for its use. The remaining steps should be performed by the owning account, and without DBA privileges. This ensures that data objects are not inadvertently created in tablespaces belonging to other databases.

Function	User	SQL Command
Allocate Storage	DBA	create tablespace ... datafile ...
Create Owner	DBA	create user ...
Create Role(s)	DBA	create role ...
Create Database	Owner	create schema
Load Data	Owner	insert into table
Grant Access	Owner	grant ... on table ... to role
Disconnect Owner	DBA	revoke CONNECT from ...

Table 5-3: Database Creation

Allocate storage. This step is performed by the DBA and creates the physical storage needed for the database. Developers shall not assume any particular disk configuration when creating data files and shall create all files in the segment's `DBS_files` subdirectory. Developers may create multiple storage areas (i.e. Oracle tablespaces) to separate different groups of data objects. Developers shall not modify the core database storage areas without permission of the DISA Chief Engineer.

Create owner. This step is performed by the DBA and creates the account or accounts that will own the data objects. Their access will be limited to the storage areas created by the segment or to public storage areas (e.g. Oracle tablespace TEMP or USERS). Owners shall not have access to system storage areas (e.g. Oracle tablespace SYSTEM). No permanent objects will be created in public storage areas by database segments. No objects will be created in system storage areas. Owners shall not have database administrator privileges.

Create roles. This step is performed by the DBA and creates the roles necessary to manage user access. Developers should match the role definitions to the access needed by applications. Developers should not grant privileges that allow users to manipulate the data objects' structure (e.g. Oracle's Alter privilege). Users should not be allowed to create their own indexes either.

Create database. This step is performed by the Owner and creates tables, views, indexes, constraints, sequences, and any other data objects that are part of the database. If the developer has defined multiple owners, a separate script should be provided for each one. No objects will be created that will be owned by the DBMS default accounts (Oracle's SYS or System, Sybase's sa) or by some other account intended to be a DBA.

Load data. This step is performed by the Owner and fills the data objects previously created. Although index and constraint creation were defined as occurring in the previous step, developers may defer them until the data load is complete to improve performance.

Grant access. This step is performed by the Owner and grants the appropriate access permissions on data objects to the roles previously defined. Grants shall not be made directly to users accounts. Grants shall not be made to general purpose users (e.g. Oracle's PUBLIC user). Only the owner or the DBA are allowed to administer grants. Other users will not be given permissions to further disseminate grants.

Disconnect Owner. The last step – revoking database connection privileges from the owner upon completion of the load process – is performed by the DBA. It ensures that users cannot connect to the database as the owner of the data, and thereby prevents users from modifying schemas, indexes, or grants. Developers shall also require the database administrators to change the password of the owner account upon completion of the database creation.

5.6.4 Database Segment Deinstall

Deinstallation has a different flavor with databases. First, data is dynamic. As users make changes to their databases, sites' data sets will diverge from each other. It is unlikely that any two operational sites will have exactly the same data at any point in time. Second, inter-database dependencies restrict the ability to remove segments modularly.

However, developers need to provide the capability to remove the application's server segment from the Database Server. This means removing the database and all traces of its presence from within the DBMS. The following steps, at a minimum, must be accomplished.

Function	User	SQL Command
Remove roles	DBA	drop role ...
Remove objects	owner	drop schema ...
Remove storage	DBA	drop tablespace ...
Remove owner	DBA	drop user ...

Table 5-4: Database DeInstall

Within the Oracle server, combining the removal of storage and of data objects by using the Oracle command ‘drop tablespace *x* including contents’ is not recommended. Developers should use the ‘drop schema’ command followed by a ‘drop tablespace’ command instead.

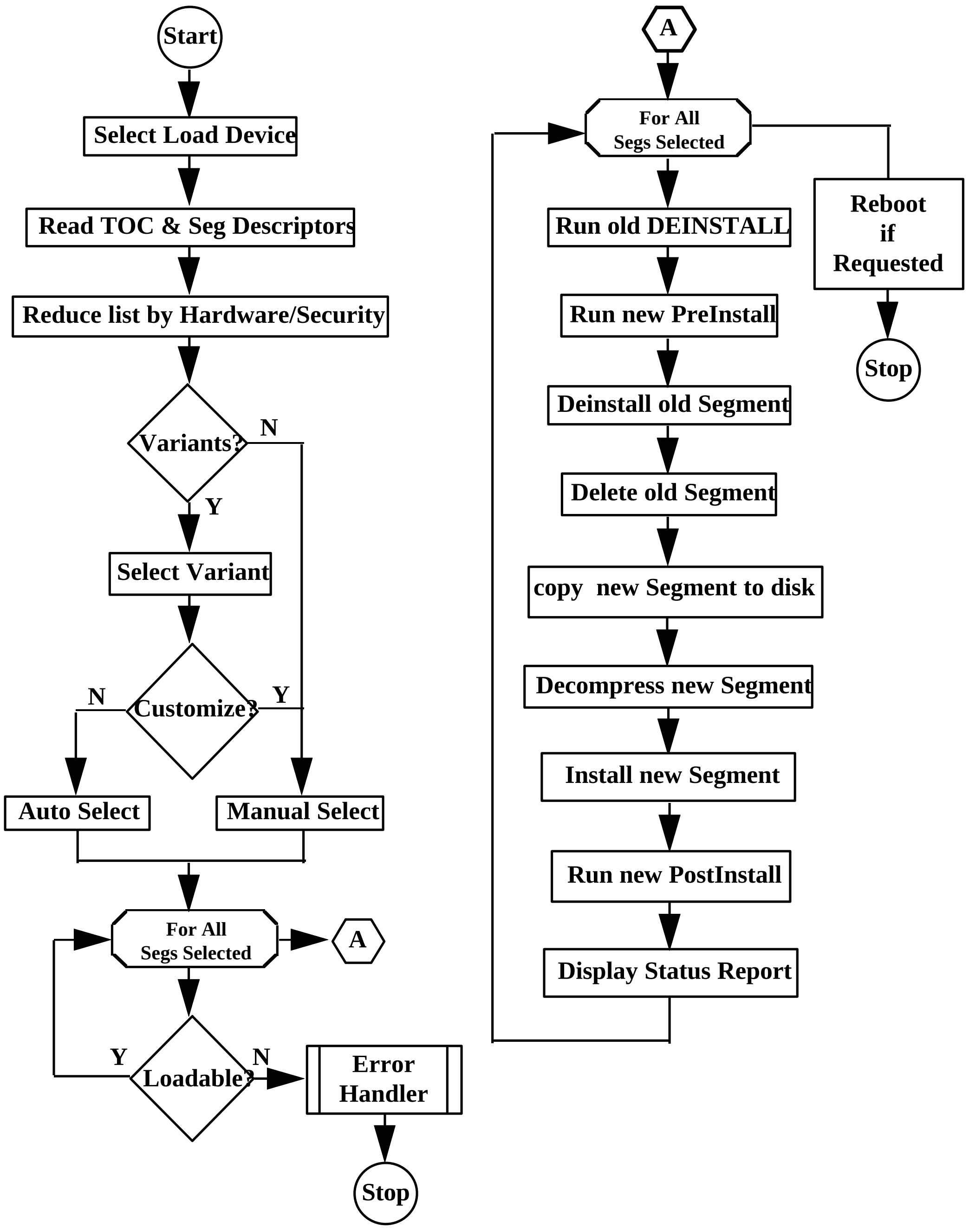

Figure 5-14: Installation Flowchart

5.7 Extending the COE

Most properly designed segments will not require any extensions to the COE, except for the need to add icons and menu items. This subsection describes some of the more commonly required extensions, and techniques for addressing less frequently encountered extensions.

5.7.1 Adding Menu Items to the Desktop

Adding menu items is usually required only when installing a software segment. Two pieces of information are required: the name of the affected account group(s) and the menu items to add. Refer to the `SegName` and `Menus` descriptors.

The installation software appends the contents of the segment's menu files to the corresponding menu files in the affected account group(s). This forms a master template in the affected account group's `data/Menus` subdirectory which is subsequently used to create operator profiles. Segments should use the `APPEND` directive in the menu files to add items. Refer to the Executive Manager API documentation for the format of menu files.

Previous COE releases included a system menu bar that was displayed at the top of the screen, just below a security banner. The COE no longer automatically provides a system menu bar. Segments which require a system menu bar must use the Executive Manager APIs to explicitly add menu items when the application initializes. Developers may only add menu items that are contained within the current user's profile. The APIs are constructed to prevent addition of menu items to the system menu bar that are not contained in the current user profile.

Segments which use a system menu bar must also use the APIs to remove their system menu bar additions when the application terminates. Refer to the *Style Guide* for guidance on when it is appropriate to use a system menu bar versus desktop icons.

Obsolete: Previous COE releases automatically provided a system menu bar. Applications must now use the Executive Manager APIs to explicitly request a system menu bar.

5.7.2 Adding Icons to the Desktop

As with menus, adding icons is usually required only for software segments. Two pieces of information are required: the name of the affected account group and the icons to add. Refer to the `SegName` and `Icons` descriptors above.

The installation software appends the contents of the segment's icon files to a master list located with affected account group(s). This forms a master template in the affected account group's `data/Icons` subdirectory which is subsequently used to create operator profiles. Refer to the Executive Manager API documentation for the format of icon files.

Refer to the *Style Guide* for guidance on when it is appropriate to use a system menu bar versus desktop icons.

5.7.3 Modifying Window Behavior

The *Style Guide* defines required window behavior for all segments. X Windows controls window behavior through a collection of resource definitions. The resource definitions consulted are as follows:

1. Files located in the directory `/usr/lib/X11/app-defaults`.
2. Files in the directory pointed to by `XAPPLRESDIR`.
3. Resources inherited from the display's root window.
4. The file `$HOME/.Xdefaults`.
5. The file pointed to by `XENVIRONMENT`.

X Windows processes the controls in the order shown, and in such a way that the last control specified overrides any preceding controls.

The COE must carefully control resources to avoid conflicts between segments. Therefore, segments shall *not* place files in directories "owned" by X Windows (e.g., `/usr/lib/X11/app-defaults`.) Instead, segments shall place their resources in the subdirectory `data/app-defaults` underneath the segment directory as shown in Figure 5-2. At install time, the installation tools create a symbolic link underneath `/h/data/local/app-defaults` to each of the files contained in the segment. For this reason, segments must use their segment prefix to name all app-defaults used in this manner.

Figure 5-2 also shows that segments may place additional fonts underneath the segment's `data/fonts` subdirectory. At install time, the installation tools create a symbolic link underneath `/h/data/local/fonts` to point to each of these files. Segments shall use their segment prefix to name font files used in this way.

The environment variables `XFONTSDIR`, `XAPPLRESDIR`, and `XENVIRONMENT` are established by the COE. Segments shall not modify their value. They are set as defined in section 5.3.

The `.Xdefaults` and `.xsession` files are extended through the `ReqrdScripts` descriptor. The installation tools use `xrdb` to merge the segment's `.Xdefaults` extensions with the affected account group, and `source` to merge in the `.xsession` additions. The installation tool performs the additions in such a way that they do not override the environment established by the affected account group.

Motif follows a similar strategy for setting resources. The following resources are consulted in the order shown:

1. Files located in `/usr/lib/X11/app-defaults/Mwm`.
2. Files located in `$HOME/Mwm`.
3. Properties inherited from the root window.
4. Properties found in `$HOME/.Xdefaults`.
5. Properties found in the file pointed to by `XENVIRONMENT`.

Segments may specify Motif resources only through a `.Xdefaults` environment extension file, or through a `Mwm` environment extension file if the affected account group supports it. Segments may *not* place files in any directory "owned" by Motif (e.g., `/usr/lib/X11/app-defaults/Mwm`), nor may segments alter the account group's `.mwmrc` resource file.

To summarize, for COE compliance:

¥ Segments shall *not* modify vendor distributed X Windows or Motif system resources (`Xdefaults`, `rgb.txt`, etc.).

¥ Segments shall *not* place files in the X or Motif distribution directories (e.g., `/usr/lib/X11/app-defaults`).

¥ Segments shall use the segment prefix to uniquely name files underneath the segment's `data/fonts` and `data/app-defaults` subdirectories.

¥ Segments shall *not* modify the COE established setting for `XAPPLRESDIR`, `XENVIRONMENT`, or `XFONTSDIR`.

¥ Segments shall *not* modify the affected account group's `.mwmrc` file.

5.7.4 Using Environment Extension Files

The `ReqrdScripts` descriptor allows extensions to the affected account group's "dot" files (`.cshrc`, `.login`, etc.). This is most frequently done to add environment variables. However, unregulated use of environment variables is detrimental to the system. The amount of space the operating system reserves for environment variables is limited, and loading a large number of segments could quickly exhaust this scare resource. Each time a process is spawned, the child process inherits environment variables from the parent. Resolving a large number of environment variables can take a significant amount of time and hence degrade system performance.

COE compliance requires adherence to the following guidelines:

¥ Do not include development environment variables in runtime environment scripts or extension files.

¥ Use short names for environment variables. Unix stores environment variable names as character strings in the environment space, so the longer the variable name, the faster environment variable space is exhausted.

¥ Reuse environment variables already defined by the COE or affected account group.

¥ When feasible and efficient, use operating system services (such as pipes and streams) or data files to communicate with other segments, or between components within the same segment.

¥ Do not use environment variables to communicate control data between components within the same segment. Use operating system services or data files.

¥ Do not define environment variables which can be derived from other environment variables. For example, to define `MYSEG_BIN` through

```
setenv MYSEG_HOME    /h/MySeg
setenv MYSEG_BIN          $MYSEG_HOME/bin
```

wastes environment variable space. The COE guarantees a predictable directory structure, and `$MYSEG_HOME/bin` can be used directly instead of `$MYSEG_BIN`.

¥ When feasible, have segment components create environment variables once they begin executing through `putenv` or through "sourcing" a file containing needed environment variables. This approach ensures that segment specific environment variables are inherited locally by a single segment, not globally by all segments.

5.7.5 Using Community Files

Community files are any files that reside outside a segment's assigned directory. (Data files owned by the segment underneath `/h/data` are considered an exception.) Most required community file modifications are handled automatically by the installation software through descriptor directory files. The `Community` descriptor is used when the installation software can not provide the modifications required.

All community file modifications are carefully scrutinized at integration time because of the potential for conflict with other segments or the runtime environment. Developers should seek guidance from the DISA Chief Engineer before modifying any COTS community files (those owned by Unix, X Windows, Motif, Oracle, Sybase, etc.).

5.7.6 Defining Background Processes

When an operator logs in, Unix uses the various "dot" files to establish a runtime environment context. The `Runxxx` program from the appropriate account group is executed to launch all processes required to complete the establishment of the runtime environment. Segments use the `Processes` descriptor file to add other background processes to the runtime environment.

The COE differentiates between four different types of processes:

Boot	Processes launched when the computer is booted or rebooted. Designate boot processes with the `$BOOT` keyword.
Background	Processes launched the first time an operator logs in after a reboot; these processes remain active in the background even after the operator logs out. Designate background process with the `$BACKGROUND` keyword.
Session	Processes launched when an operator logs in and remaining active only while the operator is logged in. Designate session processes with the `$SESSION` keyword.
Transient	Processes launched in response to operator selections from an icon or menu. Transient processes typically display a window on the screen, perform some specific function in

response to operator actions, and then terminates. In some cases, the processes spawned may stay active for the length of the session, but in all cases, transient processes are terminated by the Executive Manager when the operator logs out. Designate transient processes through the `Menus` and `Icons` descriptors.

Note: Because of the potential impact to other segments, system performance, and system integrity, boot and background processes require prior approval by the DISA Chief Engineer. Boot processes are *strongly* discouraged.

5.7.7 Reserving Disk Space

Segments frequently require additional disk space to accommodate growth over time as the system operates. For example, communications logs are empty when the system is initially installed, but will occupy space as messages are received and logged. Segments may reserve additional disk space through the `Hardware` descriptor.

The installation software keeps track of how much disk space is actually in use and how much is reserved. A segment will not be installed if the amount of space it occupies, plus any space it reserves, exceeds the amount of unreserved disk space. The installation software allows the operator to select how full the disk can be (80, 85, 90, or 95% of capacity). These safeguards are in place to avoid filling up the disk, but segments are responsible for detecting when the amount of space requested is not available.

In rare situations, segments may require space on multiple disk partitions. See the `$PARTITIONS` keyword for the `Hardware` descriptor.

5.7.8 Using Temporary Disk Space

Segments may require temporary disk space during segment installation, or during system operation. Temporary disk space may be requested during segment installation through the `$TEMPSPACE` keyword in the `Hardware` descriptor. The installation software automatically deletes all files in this temporary area when segment installation is completed.

The environment variable `TMPDIR` points to a temporary directory that may be used during system operation. However, there is a limited amount of disk space set aside for temporary storage so it must be used sparingly. A better approach is for segments to store temporary data in their own `data` subdirectory.

Segments which use TMPDIR must delete temporary files when they are no longer required. All files in this directory are automatically deleted when the system is rebooted.

5.7.9 Defining Sockets

Requests to modify the `/etc/services` file to add sockets is done through the `COEServices` descriptor file. This control point for requests to add socket names and ports helps avoid conflicts between segments. Port numbers in the range 2000-2999 are reserved for COE segments. Segments should avoid creating sockets with port numbers less than 1000 since these are generally reserved for operating system usage.

5.7.10 Adding and Deleting User Accounts

Segments are not normally allowed to create operator accounts (e.g., Unix user login accounts). Segments may create system accounts, through the `COEServices` descriptor, for the purpose of establishing file ownership. Operator accounts are normally added to the system through use of the Security Administrator application. They are customizable by security classification level, by access permissions granted or denied against application objects, and by granting or denying access to menu or icon items. The segment descriptors `AcctGroup`, `Security`, `Permissions`, `Menus`, and `Icons` provide these controls.

Figure 5-3 shows that operator accounts may be global or local. This attribute is specified when the operator account is created. If the server which contains operator accounts is down, global operator logins will be unavailable until the server is restored.

Profiles may also be global or local. This attribute is determined when the profile is created. If a global profile is not available at login time (e.g., the server is down), login proceeds but the operator is notified of the problem and the system is placed in a safe state.

Some segments require the ability to perform additional operations when a user account is created, or to perform cleanup operations when a user account is deleted. This is done by using the `$ACCTADD` and `$ACCTDEL` keywords in the `Direct` descriptor. Due to security implications, both of these keywords require prior permission from the DISA Chief Engineer.

5.7.11 Adding Network Host Table Entries

Workstation IP addresses and hostnames are site dependent. Hostnames in particular are most often selected by the site and usually can not be predicted in advance. Therefore, segments shall not include any assumptions about a workstation having a specific name or following any particular naming convention, nor make any assumptions about a specific IP address class.

Segments should rarely need to add entries to the network host table. Such entries are usually established by an operator through system administration functions. For those situations where a segment must do so, the `$HOSTS` keyword in the `Network` descriptor allows IP addresses, hostnames, and aliases to be added to the network host table. The address may be added to either the local host table, or to the DNS/NIS+ maintained host table.

Prior permission must be given by the DISA Chief Engineer to use the `$HOSTS` keyword, and permission will be granted only for COE component segments. `VerifySeg` will issue a warning for any segment which uses the `$HOSTS` keyword.

5.7.12 Registering Servers

Servers are registered with the COE through the `$SERVERS` keyword in the `Network` descriptor. Only COE component segments may register servers. Prior permission must be given by the DISA Chief Engineer to use the `$SERVERS` keyword. `VerifySeg` will issue a warning for any segment which uses the `$SERVERS` keyword, and strictly fail the segment if it is not a COE component segment.

A segment which needs to determine the location of a server may use the COEFindServer function (see Appendix C).

5.7.13 Modifying Network Configuration Files

Setting up a network requires modification of several network configuration files to set netmasks, identify subnets and routers, etc. Proper network configuration is essential for proper system operation and performance. For this reason, only COE component segments may establish network configuration parameters. This is accomplished through the `Network` descriptor file.

Prior approval from the DISA Chief Engineer is required. `VerifySeg` will issue a warning for any segment which uses the `Network` descriptor, and strictly fail the segment if it is not a COE component segment.

5.7.14 Establishing NFS Mount Points

NFS mount points are defined through the `$MOUNT` keyword in the `Network` descriptor. System performance can be seriously impacted by establishing mounted file systems. Poor design choices that result in several different mount points can create single points of failure, or result in sequencing problems when the system is loaded or rebooted. For these reasons, mount points are restricted to COE component segments.

Prior approval from the DISA Chief Engineer is required to create NFS mounted file systems. `VerifySeg` will issue a warning for any segment which uses the `$MOUNT` keyword, and will strictly fail the segment if it is not a COE component segment.

5.7.15 License Manager

The COE contains a license manager to administer COTS licenses. Vendors take a variety of approaches in how they control and administer licenses. For this reason, the techniques for automating license management are still under development and are being handled manually. Refer to the DISA Chief Engineer for further assistance in creating a segment which requires a license manager.

5.7.16 Shared Libraries

The COE strongly encourages the use of shared libraries to reduce memory requirements. Because of the different approaches taken by various legacy systems, specific techniques and direction are under development. Refer to the DISA Chief Engineer for guidance in using shared libraries, especially those associated with COTS products such as X and Motif.

5.7.17 Character Based Interface

Support for character based interfaces is provided through the `CharIF` account group. An account is established for individual users through the same process as all other accounts, but the account is noted as a character based interface account only. Operator profiles may be set up, but only those segments which support a character based interface (see the `Direct` descriptor) are accessible.

The remote user connects to the designated server through a remote login session. Once connected, the user is prompted for a login account and password. A menu of options, such as

```
0) Exit
1) AdHoc Query
2) TPFDD Edit

Enter Option:
```

is presented to the user. The option selected is executed and results are displayed on the user's remote, character based display.

5.7.18 Remote versus Local Segment Execution

Segments which are remotely launchable are designated by the `$REMOTE` keyword in the `Direct` descriptor. This feature is not currently implemented, but is reserved for future implementation. Developers are encouraged to use the `$REMOTE` keyword and design their segments to account for local versus remote execution. Thus, when this feature is fully implemented, developer segments will be position to take advantage of the capability.

5.7.19 Color Table Usage

The COE must carefully control how the color table is used to avoid objectional "false color" patterns that may appear when mouse focus changes from one window to another. The *Style Guide* gives guidance on what colors to use from a human factors perspective, but it does not provide guidance on how segments are to coordinate such usage through the COE.

This document will be expanded to include guidance for color table usage as the impact of COTS products (such as CDE) are evaluated.

5.8 Security Considerations

COE-based systems typically operate in a classified environment. Therefore, security considerations must be addressed both by the COE and the segment developer. This section describes the security implications from a runtime environment perspective. It does not address procedural issues such as proper labeling of electronic media, requirements for maintaining paper trails showing originating authority, etc.

Certain restrictions described below are a result of how Unix manages file versus directory permissions. The most specific permission (e.g., on a file) does not consistently override the least specific permission (e.g., on the file's parent directory).

This section is evolving as security policies are developed for GCCS and GCSS, and as legacy systems are migrated to the COE. Further guidance will be issued as appropriate. Refer to the DISA Chief Engineer for specific security concerns, or for guidance in segment development in addition to the information contained here.

5.8.1 Segment Packaging

Segments shall not mix classification levels within the same segment. It is permissible to create an aggregate that contains segments that are at different classification levels, but the parent segment must dominate the security level of any child segments.

Features that are not releasable to foreign nationals shall be clearly identified through documents submitted to DISA when the segment is delivered. Software and data which contain non-releasable features shall be constructed so that the features may be removed as separate segments.

All classified data shall be constructed as separate segments. Developers shall submit unclassified sample data to DISA, as a separate segment, for DISA to use during the testing process.

5.8.2 Classification Identification

All segments shall identify the segment's highest classification level in the Security descriptor. Developers shall submit documentation to DISA which clearly identifies what features are classified, and at what classification level.

5.8.3 Auditing

Segments which write audit information to the security audit log shall include the segment prefix in the output. This is required so that audit information can be traced to a specific segment.

5.8.4 Discretionary Access Controls

Developers shall construct their segments so that individual menu items and icons can be profiled through use of COE profiling software. The profiling software allows a site administrator to limit an individual operator's access to segment functions by menu item, or by icon.

5.8.5 Command Line Access

Segments shall not provide an xterm window or other access to a command line, unless prior permission is granted by the DISA Chief Engineer. Segment features should be designed and implemented in such a way that operators are not required to directly enter operating system commands. When command line access is granted, it shall not be "root" access (e.g., privileged user). Situations requiring superuser access shall require the operator to log in as root.

Segments which provide command line access shall audited entry to and exit from the command line access mode. Entry to command line access mode shall require execution of the system login process so that the user is required to enter a password. For example, the command

```
   xterm -exec login
```

will create an xterm window that requires the operator to provide a login account and password.

5.8.6 Privileged Processes

Segments shall minimize use of privileged processes (e.g., processes owned by root or executed with an effective root user id). In all cases, privileged processes shall terminate as soon as the task is completed.

5.8.7 Installation Considerations

Segments shall not require `PostInstall`, `PreInstall`, or `DEINSTALL` to run with root privileges unless permission to do so is granted by the DISA Chief Engineer.

Segments shall not alter the umask setting established by the COE.

5.8.8 File Permissions

Segments shall not contain any files directly underneath the segment's assigned directory. The reason for this restriction is that such files can be modified by an unauthorized user if the directory permissions are set to octal 777 (as the COE requires them to be for certain directories).

Segments shall not place any temporary files in the directory pointed to by `TMPDIR` unless deletion, alteration, or examination of such files by another segment or user poses no security concerns.

5.8.9 Data Directories

Segments which contain data that must have world access privileges along with data that must *not* have world shall split the data into separate directories underneath the segment's `data` directory. File permissions on the separate directories must be set to prevent unauthorized access to data files.

5.9 Database Considerations

COE-based systems are typically heavily database oriented. Database considerations are therefore of paramount importance in properly architecting and building a system. This section provides more detailed technical information on properly designing databases and database applications.

5.9.1 Database Segmentation Principles

A COE database server is provided by a COTS DBMS product. It is used in common by multiple applications. It is a services segment and part of the COE. However, different sites need varying combinations of applications and databases. As a result, databases cannot be included in the DBMS segment. Instead, these component databases are provided in a database segment established by the developer. The applications themselves are in a software segment, also established by the developer, but separate from the database segment. If the data fill for the database contains classified data, that data fill must be in a separate data segment associated with the database segment.

5.9.1.1 Database Segments

The DBMS is provided as one or more COTS segments. These segments contain the DBMS executables, the core database configuration, database administration utilities, DBMS network executables, and development tools provided by the DBMS vendor. Databases are provided as database segments. These segments contain the executables and scripts to create a database, and tools to load data into the database.

The following functional groupings are used to provide database services. The configuration of COTS segments that provide them may vary depending on the DBMS and the specific configuration chosen by DISA.

<u>DBMS Server</u>. This provides the DBMS executables, the DBMS's network services executables, and the core database. Its components are installed on the database server.

<u>DBMS Tools</u>. This provides the executables for other DBMS applications (e.g. Oracle*Forms 4.5 development tools). Its components are installed on the database server.

<u>DBMS DBA Tools</u>. This segment contains the executables for tools used by database administrators (e.g. Oracle's ServerManager). Its components are installed on the database server.

DBMS Client Services. This segment contains the client network services for the DBMS and run-time executables for other DBMS applications (e.g. Oracle*Forms 4.5 runform executable). It is installed on the network's application server. It may be installed on individual workstations.

The following specific segments are prepared by developers to provide databases within a COE-based system configuration.

Application Database Segment. This segment contains a component database. It is installed on the database server.

Application Client Segment. This application segment contains applications that access a database created by an Application Database Segment. It is installed on the network's application server or on individual workstations.

Application Data Segment. This segment contains the data fill of a component database when that data fill must be separated from the Application Database Segment. It is installed on the database server.

5.9.1.2 Database Segmentation Responsibilities

Three groups are involved in the implementation of database segments: DISA, the application developers, and the sites' database administrators. The developers and DISA work together to field databases and associated services for the DBAs to maintain. DISA provides the DBMS as part of the COE. Developers provide the component databases. Sites manage access and maintain the data. Users interact with the databases through mission applications and may, depending on the application, be responsible for the modification and maintenance of data in the databases.

5.9.1.2.1 DISA

DISA provides the core database environment in which the applications' segments will be integrated. The basic functionality provided with that core environment gets the database server ready for developers to add their databases and for the sites' database administrators to add and administer users.

The initial database contains the data dictionary, rollback segments, tools, and user and temporary tablespaces. The application servers are provided with set up with the DBMS client environment so that users need only source the environment shell script to be able to connect to the server. Finally, the initial operating system and DBMS accounts are established on the database server for the sites' database administrators.

5.9.1.2.2 Developers

Developers are responsible for providing everything associated with their application's database. Developers must define the owner account(s) for their base data objects. They must define and create the data objects within those owner accounts. Aside from the data proper, developers must determine and define the access levels and privileges that must exist for their segment's database. Database roles shall be used to implement access controls to ease the maintenance burden on the DBA.

Developers may implement specific auditing within their applications and databases, but shall not modify the system's security audits.

Developers shall provide scripts for the DBA's use to add, modify and remove users privileges.

5.9.1.2.3 Database Administrators

The System and Database Administrators at each site are responsible for creating, modifying and removing users' DBMS and UNIX accounts. For security and ease of management, a "unitary login" or single account name for each user for both the operating system and the DBMS is being adopted for COE-based systems. This means that users cannot use DBMS accounts defined by developers and that developers cannot assume the existence of any particular user accounts. It also means, as required by the system Security Policy, that database actions can be traced to the individual user. Security auditing is the responsibility of the sites' DBAs. They are implemented as each site needs using the audit features provided by the DBMS.

A DBA creates users' DBMS accounts as part of the process of granting users access to applications and their associated databases. In order for this to work properly and smoothly, the developers must provide procedures, scripts, and instructions for the DBA's use. Users' access will change over time and few users will have access to all applications. The developers' procedures must support the addition of users and the revocation of users' privileges. If an application has multiple levels of privileges, then multiple procedures must be provided.

5.9.1.3 DBMS Tuning and Customization

The core DBMS instance is configured and tuned by DISA based on the combined requirements of all developers' databases taken together. This allows the DBMS Server Segments to be reasonably independent of particular hardware configurations and ignorant of specific application sets. It is not tuned or optimized beyond that.

The final tuning of the DBMS cannot be accomplished until a complete configuration is built and it has an operational load. Developers should provide information into the tuning process, but should not make their applications dependent on particular tuning parameters. Where a non-standard parameter is required for operations, developers must provide that information to DISA so the DBMS services segment can be modified accordingly.

The developers need to communicate any design assumptions and DBMS configuration requirements which must be incorporated in the DBMS set-up. If, for example, developers need any settings in the Oracle '`initGCCS.ora`' file that are not the default settings for the current DBMS version, that information needs to be provided to DISA early in the integration process for a particular release. Based on the impact of the change, a decision can be made whether to modify the baseline server configuration or to develop a DBMS segment to accompany the application's server segment and modify the in-place database instance.

Similarly, sizing of DBMS workspace, reovery logs, and the users' temporary workspace is based on the combination of the requirements of the various applications that use DBMS services. Developers must communicate their minimum requirements for these so that the core DBMS is not set to be too small. Most of the application tools provided by DBMS vendors are incorporated in the DBMS segment in the functional category of Server Tools. To ensure that needed tools are available, developers should advise DISA what COTS tools they intend to use. When such tools are used, the developer must identify the dependency in the database application segment's `Requires` file.

Developers shall not modify the core DBMS instance's configuration. Extensions or modifications of that configuration require the specific approval of the DISA Chief Engineer.

If developers modify any of the executable tools (e.g. add User Exits to Oracle*Forms), then the modified version of the tool does not reside with the core database services, but becomes a part of the application's client segment. This prevents conflicts among different modified versions of a core function. The maintenance of that modified tool also becomes the responsibility of the developers.

5.9.2 Database Inter-Segment Dependencies

A key objective of the segmentation approach is to limit the interdependencies among segments. Ideally, database segments should not create data objects in any other schema or own data objects that are dependent on other schemas.

However, one purpose in having a Database Server is to limit data redundancy and provide common shared data sets. This means that there will usually be some dependencies among the databases in the federation. This section addresses the management of such dependencies.

The following principles apply when inter-database dependencies exist:

¥ The database schema within a segment that will own the parent object will create that object.
¥ The database schema within a segment that will own the child (dependent) object will create that object.
¥ Database schemas with inter-database dependencies will strive to keep those dependencies in segments separate from the non-dependent portions of the schema.
¥ Schemas retain their autonomy. The developer of a dependency is responsible for maintaining that dependency should other developers change their database schemas.

Database Segments shall not make modifications to another segment’s database. If a schema needs to create data objects in some schema belonging to another segment, those objects will be placed in a database segment that modifies the segment that will own the objects. Developers should try to avoid creating additional indexes on another segment’s tables because of the performance problems they can cause.

Developers will not modify the schema of another segment’s database. If changes to table or column definitions are needed, they must be effected by the developer of the database.

When dependencies exist they will be documented in the `Requires` file of the `SegDescrip` directory.

The following example illustrates how dependencies are to be created and managed. The developers of database B need to attach a trigger to a table in database A. This trigger will feed data from A to B every time that table is modified. Rather than include the trigger as part of B’s Application Server Segment, it is put into a separate Segment on Database A. The Inter-database Segment is dependent on the prior installation of both database segments and is so labeled in its `Requires` File.

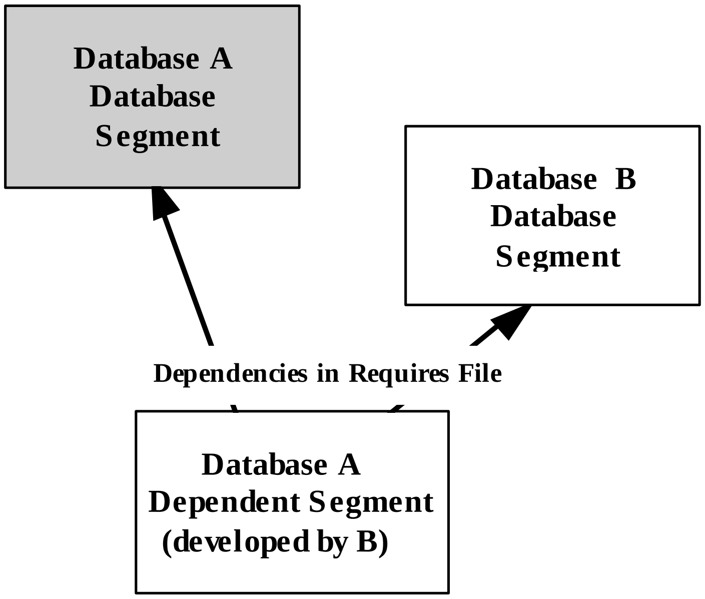

Figure 5-15 Inter-Database Dependencies